# Data-driven flight path rerouting during adverse weather

Design and development of a passenger-centric model and framework for alternative flight path generation using nature inspired techniques

Babatope Samuel AYO

Submitted for the Degree of
Doctor of Philosophy

Faculty of Engineering and Informatics

University of Bradford

2018

# Abstract


**DATA-DRIVEN FLIGHT PATH REROUTING DURING ADVERSE WEATHER**

Design and development of a passenger-centric model and framework for alternative flight path generation using nature inspired techniques



**Keywords**

Flight path, adverse weather, shortest path, rerouting, passenger inconvenience, genetic algorithm, heuristics, optimisation



A major factor that negatively impacts flight operations globally is adverse weather. To reduce the impact of adverse weather, avoidance procedures such as finding an alternative flight path can usually be carried out. However, such procedures usually introduce extra costs such as flight delay. Hence, there exists a need for alternative flight paths that efficiently avoid adverse weather regions while minimising costs.

Existing weather avoidance methods used techniques, such as Dijkstra's and artificial potential field algorithms that do not scale adequately and have poor real time performance. They do not adequately consider the impact of weather and its avoidance on passengers.

The contributions of this work include a new development of an improved integrated model for weather avoidance, that addressed the impact of weather on passengers by defining a corresponding cost metric. The model simultaneously considered other costs such as flight delay and fuel burn costs.

A genetic algorithm (GA)-based rerouting technique that generates optimised alternative flight paths was proposed. The technique used a modified mutation strategy to improve global search. A discrete firefly algorithm-based rerouting method was also developed to improve rerouting efficiency. A data framework and simulation platform that integrated aeronautical, weather and flight data into the avoidance process was developed. Results show that the developed algorithms and model produced flight paths that had lower total costs compared with existing techniques. The proposed algorithms had adequate rerouting performance in complex airspace scenarios. The developed system also adequately avoided the paths of multiple aircraft in the considered airspace.

## Acknowledgements

All praise and thanks be to God through Jesus Christ, ‘in whom we live, move and have our being’.

I sincerely thank my supervisors, Professor Y.F. Hu and Dr J.P. Li, for their expert supervision, support and patience during this doctoral research.

I am equally grateful to Professor P. Pillai, my associate supervisor until he left the University of Bradford, for his support and expert supervision.

My very deep gratitude goes to Dad and Mum for their loving, sacrificial and unflinching support, advice, encouragement and prayers – without which the start and completion of the research would have been impossible.

I am very grateful to my dear siblings for their support, understanding and prayers throughout the course.

My deep appreciation goes to the pastors and members of Queensbury Life Church for their loving support and prayers.

I would like to appreciate the support and advice of past and present staff and students across the University of Bradford, including those of the Future Ubiquitous Networks Laboratory and J.B. Priestley Library.

# Dedication

*To Dad and Mum*

*To under-privileged children everywhere*

# Table of Contents

# List of Figures

## Abbreviations

| | |
|---|---|
| AC | Ant Colony |
| ACO | Ant Colony Optimisation |
| ACS | Ant Colony System |
| ADS-B | Automatic Dependent Surveillance-Broadcast (ADS-B) |
| AIP | Aeronautical Information Publication |
| AIREPS | Aircraft Reports |
| AIRM | ATM Information Reference Model (AIRM) |
| AIRMET | Airmen's Meteorological Information |
| AIXM | Aeronautical Information Exchange Model |
| AMQP | Advanced Message Queuing Protocol |
| AOC | Aircraft Operational Communication |
| AOP | Autonomous Operations Planner |
| APC | Aircraft Passenger Communications |
| API | Application Programming Interface |
| ARTCC | Air Route Traffic Control Centre (ARTCC) |
| ASDI | Airline Situation Display to Industry |
| ASN | Abstract Syntax Notation |
| ATC | Air Traffic Control |
| ATM | Air Traffic Management |
| ATS | Air Traffic Service |
| BADA | Base of aircraft Data |
| BER | Basic Encoding Rules |
| BPEL | Business Process Execution Language |
| BSI | British Standards Institution |

| | |
|---|---|
| CAA | Civil Aviation Authority |
| CAT | Clear Air Turbulence |
| CDR | Coded Departure Routes |
| CICTT | Commercial Aviation Safety Team/ICAO Common Taxonomy Team |
| CIP | Current and Forecast Icing Product |
| CIWS | Corridor Integrated Weather System |
| CLR | Common Language Runtime |
| CORBA | Common Object Request Broker Architecture |
| CTAS | Centre/TRACON Automation System |
| CWAM | Convective Weather Avoidance Model |
| DAR | Dynamic Arrival Routes |
| DCOM | Distributed Component Object Model |
| DCPS | Data-Centric Publish-Subscribe |
| DDS | Data Distribution Service for Real Time Systems |
| DFA17 | Discrete Firefly Algorithm (proposed) |
| DFW | Dallas/Fort Worth International Airport (ICAO code) |
| DLRL | Data Local Reconstruction Layer |
| DNS | Domain Name System |
| DOF | Degree Of Freedom |
| DSP-M | Improved Dijkstra's Shortest Path algorithm |
| DWR | Dynamic Weather Routes |
| EEA | European Environment Agency |
| EE | Enterprise Edition (Java programming platform) |
| EFPT | En route Flow Planning Tool |

| | |
|---|---|
| EGBB | Birmingham International Airport |
| EGCC | Manchester Airport (ICAO code). |
| EGJJ | Jersey Airport (ICAO code) |
| EGKK | London Gatwick Airport (ICAO code) |
| EGLL | Edinburgh Airport (ICAO code) |
| EGPF | Glasgow International Airport (ICAO code) |
| EGPH | London Heathrow Airport (ICAO code) |
| EJB | Enterprise Java Beans |
| EUROCONTROL | European Organisation for the Safety of Air Navigation |
| FA | Firefly Algorithm |
| FAA | Federal Aviation Administration |
| FCFS | First Come First Serve |
| FIP | Forecast Icing Product |
| FIXM | Flight Information Exchange Model |
| FIXS | Flight Object Schema |
| FL | Flight Level |
| FMS | Flight Management System |
| GA | Genetic Algorithm |
| GAARM | Genetic Algorithm with Replacement Mutation strategy |
| GAIM16 | Genetic Algorithm with Improved Mutation (proposed) |
| GB | Gigabyte |
| GML | Geography Markup Language |
| GMT | Greenwich Mean Time |
| GOES | Geostationary Operational Environmental Satellite |
| HMI | Human Machine Interface |

| | |
|---|---|
| HTTP | Hypertext Transfer Protocol |
| IAF | Initial Approach Fix |
| IATA | International Air Transport Association |
| IBM | International Business Machines Corporation |
| ICAO | International Civil Aviation Organisation |
| IDL | Interface Definition Language |
| IEEE | Institute of Electrical and Electronics Engineers |
| IFR | Instrument Flight Rules |
| ILS | Instrument Landing System |
| IP | Internet Protocol |
| ISO | International Organization for Standardization |
| ITU-T | International Telecommunication Union - Telecommunication Standardization Sector |
| ITWS | Integrated Terminal Weather System |
| IWXXM | ICAO Meteorologic Information Exchange Model |
| JAX-RS | Java API for RESTful Web Services |
| JAX-WS | Java API for XML Web Services |
| JMS | Java Message Service |
| JPEG | Joint Photographic Experts Group format |
| JSON | JavaScript Object Notation |
| LNAV | Lateral Navigation |
| LP | Linear Programming |
| METAR | Meteorological Terminal Air Report |
| METCE | Modèle pour l'Échange des Informations sur le Temps, le Climate et l'Éau (alternative term: METeorological Community Exchange model) |

| | |
|---|---|
| MIME | Multipurpose Internet Mail Extensions |
| MOD | Modified routes |
| MOGA | Multi-Objective Genetic Algorithm |
| MOM | Message-Oriented Middleware |
| MQ | Message Queue |
| MQTT | Message Queuing Telemetry Transport |
| MWO | Meteorological Watch Offices |
| NATS | National Air Traffic Services |
| NM | Nautical Mile |
| NOAA | National Oceanic and Atmospheric Administration |
| NSGA-II | Non-Dominated Sorting Genetic Algorithm II |
| OASIS | Organization for the Advancement of Structured Information Standards |
| OGC | Open Geospatial Consortium |
| OMG | Object Management Group |
| OO | Object-Oriented |
| OPMET | Operational meteorological information |
| OS | Operating System |
| OSI | Open Systems Interconnection |
| PBGA | Pattern-Based Genetic Algorithm |
| PER | Packed Encoding Rules |
| PIF | Passenger Inconvenience Factor |
| PIM | Platform Independent Model |
| PNG | Portable Network Graphics format |
| POSIX | Portable Operating System Interface |

| | |
|---|---|
| PSM | Platform Specific Model |
| PSO | Particle Swarm Optimisation |
| QoS | Quality of Service |
| REST | Representational State Transfer |
| RMI | Remote Method Invocation |
| RPC | Remote Procedure Call |
| RPK | Revenue Passenger Kilometres |
| RTI | Real Time Innovations |
| RTPS | Real time Publish Subscribe protocol |
| SA | Simulated Annealing |
| SASL | Simple Authentication and Security Layer |
| SCTP | Stream Control Transmission Protocol |
| SE | Standard Edition (Java programming platform) |
| SEDP | Simple Endpoint Discovery Protocol |
| SESAR JU | Single European Sky ATM Research Joint Undertaking |
| SID | Standard Instrument Departure |
| SIGMET | Significant metrological report |
| SMTP | Simple Mail Transfer Protocol |
| SOA | Service-Oriented Architecture |
| SOAP | Simple Object Access Protocol (originally) |
| SOM | Service-Oriented Middleware |
| SPDP | Simple Participant Discovery Protocol |
| SPECI | Special Weather Report |
| SRD | Standard Routes Document |
| SSR | Secondary Surveillance Radar |

| | |
|---|---|
| SWIM | System Wide Information Management |
| SWIM-SUIT | SWIM-Supported by Innovative Technologies |
| TAC | Traditional Alphanumeric Code |
| TAF | Terminal Aerodrome Forecast |
| TCP | Transmission Control Protocol |
| TLV | Type, Length and Value |
| TRACON | Terminal Radar Approach Control |
| UDDI | Universal Description, Discovery, and Integration |
| UDP | User Datagram Protocol |
| UML | Unified Modeling Language |
| VFR | Visual Flight Rules |
| VIL | Vertical Integrated Liquid |
| VNAV | Vertical Navigation |
| VOR | Very High Frequency (VHF) Omni-Directional Range |
| WADL | Web Application Description Language |
| WFS | Web Feature Services |
| WMO | World Meteorological Organization |
| WMS | Web Map Service |
| WS | Web Services |
| WSDL | Web Service Description Language |
| WXXM | Weather Information Exchange Model |
| WXXS | Weather Information Exchange Schema |
| XER | XML Encoding Rules |
| XML | Extensible Markup Language |

# List of Variables

| | |
|---|---|
| $\Delta J$ | Change in cost |
| $\Delta T$ | Change in temperature |
| *AF* | Altitude change factor |
| *attempts* | Number of attempts to remove loop in route |
| *attempts2* | Number of attempts to repair disjoint route |
| *C(n)* | Cost function in an A* algorithm |
| $C_1$, ,…, $C_c$ | Constraints to optimisation problem |
| $c_{ab}^{T}$ | Decision variable for flight duration, that is equal to 1 if the route passes through link (*a, b*) and 0 otherwise |
| $C_{FL}$ | Cost per flight level change |
| *child1,child2* | Children routes generated after crossover in a GA |
| $C_m$ | Unit cost associated with the type of a weather cell |
| $C_{Mc}$ | Cost per missed connection |
| *crs_pt1, crs_pt2* | Loci (positions) of randomly select a waypoint in $C_w$ in a GA |
| $C_w$ | list of waypoints common to both selected parents in a GA |
| $C_\delta$ | Unit cost of delay |
| $d_{ff}$ | Distance between a pair of fireflies |
| *d(r)* | Network distance of a node *r* |
| *DF* | Estimated distance from a given node to exit point |
| $D_l(a,b)$ | Leg distance between any two points *a* and *b* on a route |
| $D_n$ | Flight distance of reroute |
| $D_p$ *(i,j)* | Distance between aircraft *i* and *j* located at |
| $D_{sep}$ | Minimum separation distance between aircraft |

| | |
|---|---|
| $D_w(i,w)$ | Distance of separation from the weather phenomenon for aircraft *i* |
| $D_{w(sep)i}$ | Minimum distance of separation from the weather phenomenon allowed for aircraft of type *i* |
| *e* | An edge in network |
| *E* | Set of links between waypoints |
| $E_R$ | The earth's radius |
| $F_{Bm}$ | Fuel costs per passenger *for* a distance $D_m$ |
| *Fireflyi, Fireflyj* | Any two fireflies in an FA |
| $F_t$ | Fitness |
| $F_{Ti}$ | Total flight time |
| *fx* | Sum of exit times for integer programming method |
| *g* | Iteration number in an FA |
| *g(n)* | Distance from a current node n to start node |
| *h(n)* | Distance from a current node *n* to destination node |
| *HF* | Heading change factor |
| $h_{ij}$ | Distance between *Fireflyi* and *Fireflyj* |
| *i, j* | Any aircraft under consideration |
| *Ii* | Light intensities of fireflies in an FA |
| $I_m$ | Intensity of the $m^{th}$ weather cell |
| $J_1 \ldots J_n$ | Costs associated with a link |
| $J_{AOC}$ | Aircraft additional operating costs |
| $J_{FL}$ | Total flight level change cost |
| $J_{FL}$ | Flight level change cost |
| $J_m$ | Weather impact cost |

| | |
|---|---|
| $J_{Mc}$ | Total connection cost |
| $J_{Nc}$ | Cost of number of connections |
| $J_{PIF}$ | Passenger inconvenience factor |
| $J_t$ | Total costs |
| $J_{to}$ | Overall cost for a set of multiple aircraft |
| $J_\delta$ | Flight delay costs |
| $k$ | Iteration number in simulated annealing |
| $k_f$ | Unit fuel cost |
| $l_{ab}$ | A flight link between any two waypoints *a* and *b* |
| $L_{gb}$ | Length of the best tour in ant colony optimisation |
| $m$ | Number of ants in ant colony optimisation, or number of movements of a firefly towards a brighter firefly in an FA |
| *max_attempt1* | Maximum number of attempts to remove loop in route |
| *max_attempt2* | Maximum number of attempts to repair disjoint route |
| $n$ | Number of fireflies in an FA |
| $N_{FL}$ | Number of flight level changes |
| $Ni$ | Number of aircraft considered |
| $n_k(t)$ | Number of aircraft in the $k^{th}$ sector at time $t$ |
| $n_{kmax}(t)$ | Maximum capacity of the $k^{th}$ sector at time $t$ |
| $nl$ | Number of links on a route |
| $N_m$ | Total number of weather cells |
| $N_{Mc}$ | Number of missed connections |
| $N_p$ | Total number of passengers on aircraft |
| $N_p$ | Total number of passengers on aircraft |
| $N_{pi}$ | Total number of passengers on an aircraft *i* |

| | |
|---|---|
| $P$ | initial node |
| $P(\Delta C)$ | Probability of accepting new solution in simulated annealing |
| *parent1, parent2* | Selected parents in genetic algorithms |
| $Q$ | end node |
| $R^2$ | Coefficient of determination |
| $R_D$ | Intermediary node in a shortest path |
| $R_w$ | Coverage radius of weather phenomenon |
| $R_w$ | Radius of adverse weather region |
| *s, d* | Start and destination waypoints of a route respectively |
| $T_k$ | Temperature at the *k*th iteration in simulated annealing |
| $T_o$ | User-defined initial temperature in simulated annealing |
| $t_w$ | Estimated time of arrival at waypoint *w* |
| *(u, v)* | Nodes in graph (for Dijkstra's algorithm) |
| *V* | Set of the waypoints and airports (nodes) in the airspace. |
| *(v, w)* | Edge between nodes *v* and *w* in network |
| $v_l(a, b)$ | Average velocity of the aircraft during flight leg between *a* and *b* |
| $v_l(t)$ | Aircraft speed |
| $v_{max}(t)$ | Maximum aircraft speed |
| $v_{min}(t)$ | Minimum aircraft speed |
| $v_{wl}(t)$ | Aircraft speed in the weather phenomenon |
| $v_{wmax}(t)$ | Maximum possible speed in the weather phenomenon |
| $w(u, v)$ | Mapping specifying weights between two nodes *u* and *v* of a graph (Dijkstra's algorithm) |
| *WF* | Weather severity factor |

| | |
|---|---|
| $w_{FL}$ | Weight associated with the flight level change cost |
| $w_m$ | Weight of the weather impact cost |
| $w_{Nc}$ | Weight assigned to connection cost |
| $w_\delta$ | Weight associated with the delay cost |
| $x$ | Flight solution |
| $(x_a, y_a)$ | Latitude and longitude of waypoint $a$ |
| $x_f$ | Flow on an edge $(v,w)$ of network |
| $(x_i, y_i, z_i)$ | Cartesian coordinates of an aircraft $i$ |
| $(x_j, y_j, z_j)$ | Cartesian coordinates of an aircraft $j$ |
| $x_p$ | New solution after perturbation in simulated annealing |
| $(x_w, y_w, z_w)$ | Cartesian coordinates of centre of weather phenomenon |
| $y$ | Total fuel used in kilogrammes for a flight |
| $\alpha$ | Pheromone decay parameter in ant colony optimisation |
| $\beta$ | Parameter that determines relative importance given to pheromone information in ant colony optimisation |
| $\gamma$ | Light absorption parameter in an FA |
| $\eta(r,s)$ | Visibility parameter on a link $(r,s)$ in ant colony optimisation |
| $\lambda_a, \lambda_b$ | Longitudes of points $a$ and $b$ |
| $\tau$ | Pheromone concentration in ant colony optimisation |
| $\varphi_a, \varphi_b$ | Latitudes of points $a$ and $b$ |
| $\delta$ | Flight delay |

# Chapter 1 : INTRODUCTION

## 1.1 Background

Global air traffic is projected to grow considerably in coming years (Lim and Zhong 2017). By 2032, global passenger traffic demand is forecasted to increase to nearly 14 trillion revenue passenger kilometres (RPK), compared with less than 5 trillion RPK in 2012 (ICAO 2016b). The corresponding compound annual growth rate is expected to be 4.6%. This growth in air traffic will put substantial pressure on air traffic systems, crew and airline industry.

Such increased demand could lead to increased delays and increased airline costs. Simultaneously, stringent safety requirements need to be met and maintained. Hence, there is a need to improve existing air traffic and airline operations to ensure that they are efficient and robust to growth and disturbances.

A major factor that affects flight timeliness is weather phenomena (EUROCONTROL 2016). Adverse weather phenomena include fog, thunderstorm, volcanic ash, hail, and turbulence. Adverse weather has serious impact on aircraft safety. For example, adverse weather can cause structural damage to aircraft and lead to engine failure or performance impairment (FAA 2013b).

Adverse weather phenomena, such as icing, can affect the aerodynamic performance of the aircraft (FAA 2016a). This can lead to reduced fuel efficiency. Some hazardous weather, such as very strong winds, can impair the manoeuvrability of an aircraft by its crew. In addition, to maintain safety levels, air traffic control might reduce the number of flights landing or taking off from airports. This could lead to flight delays and congestions. Passengers are also impacted by some adverse weather. For example, sudden severe turbulence can lead to passenger discomfort or injury. Other impact on passengers include flight delays and missed connections.

As demand for air transport increases, there is the requirement to reduce aircraft emissions and noise (Park and O'Kelly 2014). This means that air transport

operations need to be efficient to meet crucial emissions targets. Therefore, techniques such as improved rerouting would contribute towards achieving emission reductions.

An important requirement for efficient air operations is the timely availability of reliable data. SWIM is a global information infrastructure that enables the seamless sharing of air transport data, such as aeronautical data, flight data, and weather data (ICAO 2017). SWIM standards support the exchange of weather data using IWXXM, an XML and GML-based format. Similarly, flight and aeronautical information can be exchanged using the AIXM and FIXM formats respectively.

In the future, aviation data providers would need to produce data like aviation weather reports in such globally interoperable formats (WMO 2016b). By using service-oriented architectures (SOA), SWIM standards ensure loose coupling between data producers and users. This means that system implementations are relatively independent of system interfaces. In addition, the impact of changing underlying technologies is reduced. There exists the need for aeronautical applications that can adequately make use of SWIM facilities to deliver innovative real-time solutions. Therefore, in the proposed solution in this research, support for SWIM data access and utilisation was included.

An important weather avoidance method is the rerouting and modification of the flight path of an aircraft (Wang and Yang 2013). Existing techniques for rerouting aircraft include artificial potential field algorithms (Xu *et al.* 2010; Zhiyang and Tao 2017), simulated annealing (Taylor and Wanke 2010; Chaimatanan *et al.* 2014; Dhief *et al.* 2017) and dynamic programming (Taylor and Wanke 2009). A major challenge is that many of the methods do not adequately scale to real-time scenarios with considerable complexity and number of waypoints. In addition, they did not adequately consider the impact of such paths on passengers.

To reduce costs to airlines and improve airspace utilisation, it is crucial that more efficient flight rerouting systems are developed. This is especially important because of the projected increase in global air traffic and required reductions in aircraft emissions.

## 1.2 Aim and objectives

The aim of this research is to develop an improved model and framework to enable the efficient rerouting of flight paths during adverse weather, in order to minimise the cost of flight rerouting. The resulting model and framework should also ensure that the impact on passengers is minimised.

The objectives of the work are:

1. To develop an improved cost model for the efficient generation of alternative flight paths during adverse weather;
2. To minimise impact of such flight path rerouting on passengers;
3. To develop improved algorithms that enable adequate path rerouting during adverse weather avoidance;
4. To develop a framework that allows the improved integration of aeronautical, flight and weather data in the weather avoidance process;
5. To provide support for the use of middleware by the developed framework to ensure seamless acquisition of input data;
6. To evaluate the performance of the proposed algorithms in terms of route costs.

## 1.3 Innovations and contributions

The contributions of this research include the following.

1. **A passenger-centric cost model for adverse weather avoidance for flights**

   A major problem with prior research in adverse weather avoidance was that they did not adequately consider the impact of flight path reroutes on passengers. This work addressed this problem by proposing a cost model that determines and minimises the impact of alternative flight path on passengers. The model considered component factors such as flight delays, weather impact, missed connections, and flight level changes. In addition, the developed model considered operational costs, such as fuel costs.
2. **An improved genetic algorithm-based rerouting technique for generation of alternative flight paths**

Modified genetic algorithm-based techniques were proposed for generating alternative flight paths during adverse weather. An enhanced mutation process for the algorithms was proposed. Algorithms developed by previous researchers often suffered from premature convergence, especially when applied to large and complex scenarios. This work developed a GA-based algorithm with an improved mutation technique to solve the problems of premature convergence and entrapment in local optima. This algorithm was developed to be suitable for short-term and in-flight path rerouting.

3. **A discrete firefly-based algorithm for flight path rerouting**

   Firefly algorithms have the advantage of relatively fewer tuning parameters. Also, by using swarm-based approaches, firefly-based algorithms can improve the search for global solutions. This work developed a rerouting method derived from the modification of the classic firefly algorithm, such that the method is suitable for discrete and network-based airspace models.

4. **Development of an integrated framework for weather avoidance**

   A major challenge of adverse weather avoidance procedures is the complexity and large amount of information that needs to be processed. To tackle this challenge, a framework has been developed that integrated aeronautical, weather, flight and surveillance data into the avoidance process. This is required to ensure that efficient alternative paths are generated. Moreover, the framework provided support for the acquisition of the relevant data using the SWIM global aviation data infrastructure.

5. **A simulation and research platform for adverse weather avoidance**

   Another challenge in flight weather avoidance research is the lack of suitable open research tools. This work developed a new simulation platform that enables the rapid design, developing and testing of re-routing techniques. The platform was developed in MATLAB to enable ease of use by end-users. The platform supports the visualisation of rerouting solutions and saving to file of simulation data for further analyses.

## 1.4 Publications

1. Ayo, B. S., Hu, Y. F. and Li, J. P. (2017). Adverse weather avoidance considering flight level changes. Proceedings of the Seventh International Conference on Innovative Computing Technology (INTECH). 16-18 Aug. 2017. This covered part of the work in Chapters 4 and 6.
2. Ayo, B. S. (2017). An improved genetic algorithm for flight path re-routes with reduced passenger impact. Journal of Computer and Communications 5 (07), 65-75, 2017. This covered work in Chapters 2, 4, 5 and 6.
3. Ayo, B. S., Hu, Y. F. and Li, J. P. (2017). A genetic algorithm with improved mutation for shortest path problems. Presented at the 1st Annual Innovative Engineering Research Conference (AIERC). This covered work in Chapters 4 and 5.
4. Ayo, B. S., Hu, Y. F. and Li, J. P. (2018). Flight Parout: A simulation platform for intelligent flight path reroutes for adverse weather, 37th AIAA/IEEE Digital Avionics Systems Conference (DASC), (accepted abstract). This presented work in Chapters 1, 2, 5 and 7.

## 1.5 Structure of the thesis

The structure of the thesis is as follows. Chapter 1 gives the background, aims and objectives of the research. Chapter 2 presents a review on adverse weather avoidance for aircraft. The chapter presents the types of adverse weather, weather data services and flight phases. The impact of adverse weather is also considered. Related work and techniques for adverse weather avoidance are discussed in the chapter.

Chapter 3 discusses air transport data and enabling technologies. Such data include aeronautical, weather and flight data. Applicable middleware classes, messaging and data formats are considered in the section. The chapter also discusses the SWIM global data framework for air transport.

Chapter 4 presents the proposed cost model for adverse weather avoidance problem. The chapter defines the problem and relevant assumptions. The chapter also discusses the constraints of the model. The derivation of each cost

component, such as flight delay costs, missed connections costs and weather impact costs is presented in the chapter.

Chapter 5 presents the proposed genetic and firefly algorithms for adverse weather avoidance. A framework for the avoidance process is also presented. The chapter discusses the components of the GA-based algorithms such as the generation of initial population, computing of fitness function, selection, mutation process, repair function. A discrete firefly algorithm for the rerouting during adverse weather is also presented.

Chapter 6 presents and discusses the results of simulations using the proposed model and algorithms. The effects of various parameters such as mutation rate, population size, and number of generations for a benchmark network chapter are presented and discussed. The section also looks at the effect of multiple regions of adverse weather. A grid-based approach is also considered in the chapter, along with the effects of multiple weather regions and aircraft avoidance. The chapter also considers the effects of flight level changes on the rerouting process.

Chapter 7 presents and discusses results for a case study. The scenario considers adverse weather avoidance in UK airspace. Chapter 8 concludes the thesis and summaries its key contributions. The chapter also identifies areas for future research.

# Chapter 2 : ADVERSE WEATHER AVOIDANCE FOR AIRCRAFT

## 2.1 Introduction

Weather has considerable influence on air traffic. In fact, weather is one of the major causes of flight delays (EUROCONTROL 2016). Adverse weather has serious impacts on passengers and aircraft safety. Therefore, affected flights often have to be rerouted to avoid such adverse weather, which may come up at short notice. However, the weather conflict avoidance process has to be done as efficiently and optimally as possible, in order to minimise fuel, travel distance and other costs.

A number of previous researchers had proposed solutions to the problem of rerouting aircraft to avoid adverse weather. These techniques include potential field model (Xu *et al.* 2010) and dynamic programming (Taylor and Wanke 2009). However, these techniques are unable to perform satisfactorily for large-scale scenarios involving large number of enroute aircrafts and constraints.

To tackle this challenge, metaheuristic techniques have been proposed and provide plausible alternatives in good time (Stewart *et al.* 2012). It is very difficult for these techniques to obtain optimal solutions for a large-scale problem due to the search spaces that are too wide. These metaheuristics include simulated annealing (SA), genetic algorithms (GA), Particle Swarm Optimisation (PSO) and ant colony algorithms (AC).

However, the passenger inconvenience factor was not considered in the existing rerouting models. This work intends to improve the work of other researchers by considering this factor, along with minimising the effects on the schedules of other aircrafts. In addition, an improved GA-based technique was used to obtain flight paths that reduce weather impact on flights.

## 2.2 Problem statement

The problem considered is to find a short-term trajectory that dynamically avoids adverse weather and minimises costs. Adverse weather phenomena include thunderstorms, turbulence, high-speed winds and reduced visibility (Krozel and Murphy Jr 2007).

The weather avoidance process is divided into weather conflict detection and resolution phases (Hauf *et al.* 2013). During the detection stage, the route segment and space under consideration is scanned and compared with forecasts or measurements from weather information providers and on-board equipment (like weather radar). A weather conflict is determined if the flight path comes within a given distance of a weather cell. This distance is the minimum separation distance $D_{sep}$ the aircraft must keep from a weather phenomenon (Fig. 2.1). For instance, the recommended value is 10 nautical miles (NM) for thunderstorms (NATS 2010)

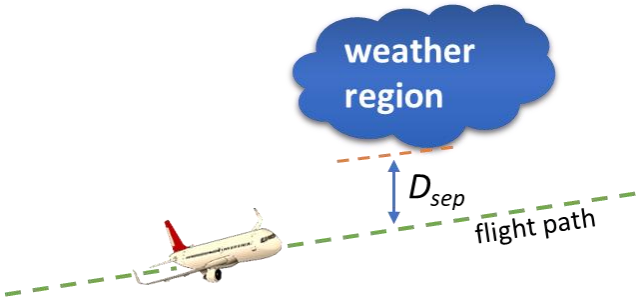


Fig. 2.1: Determination of weather conflict

The resolution phase provides alternate paths that avoids the weather phenomena. To avoid adverse weather, the aircraft executes manoeuvres such as turning left or right, as well as changing its velocity or altitude. Other less attractive corrective actions include ground and air holding. This is because of the fuel and time costs involved. In ground holding, the aircraft is delayed before taking off. For air holding, on the other hand, the aircraft flies in a circle pattern.

Consider an aircraft with an agreed route through a given airspace affected by weather cells. The weather cells are assumed to be static, with dimensions corresponding to the coverage of the adverse weather. In this work, the cells are approximated by circles with constant radii. The 4D trajectory of the aircraft is defined by its latitude ($i$), longitude ($j$), altitude ($k$) and time ($t$). The portion of the airspace within the look-ahead time is represented by a 3D hyper-rectangular shape. The time dimension is considered by taking snapshots of the 3D shape at sampling intervals $t_s$ (Nguyen *et al.* 2007; Chaimatanan *et al.* 2014), see Fig. 2.2.

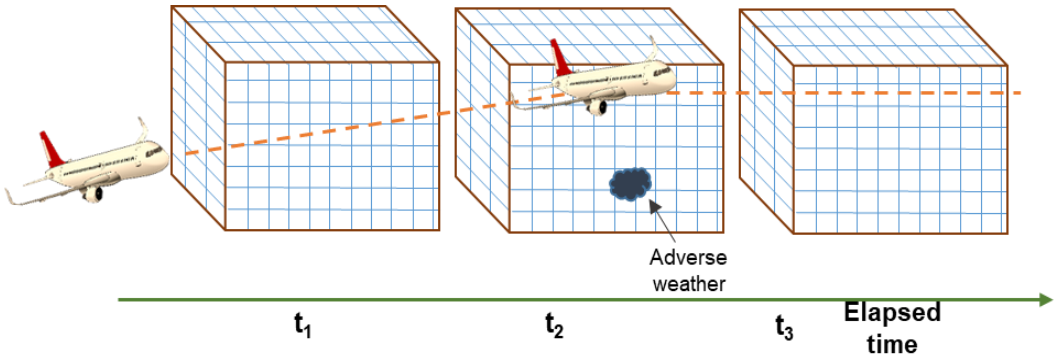


Fig. 2.2: 4D trajectory showing avoidance of adverse weather

Given a starting point *x* (before weather conflict avoidance) and an exit point *y* (after weather avoidance), the aim is to minimise a set of costs and avoid adverse weather, subject to constraints. For each aircraft, this could be represented as (Zhang *et al.* 2012):

$$\text{Minimise } \{J_1, J_2, \ldots, J_s\} \qquad (2.1)$$

$$\text{subject to } \{C_1, C_2, \ldots, C_t\}$$

where each of $J_1, J_2, \ldots, J_s$ represents each of the costs considered and $C_1, C_2, \ldots, C_t$ refers to the constraints on feasible solutions. Weather avoidance is formulated as a constraint, such that the distance from a weather cell is more than a given threshold. In the case of thunderstorms, the separation threshold is at least 10 nautical miles (NATS 2010).

An important cost to be considered is geographical distance. However, the shortest geographical distance is not always the most efficient or fastest, due to factors such as destination or sector congestion. Low fuel consumption is also desirable to reduce emissions.

Constraints refer to the operational and aircraft limitations that determine if a given solution is feasible or executable. These include aircraft characteristics (such as maximum turn angle, minimum/maximum altitude or speed) and sector occupancy.

## 2.3 Flight phases

Adverse weather has different forms of impact, depending on what flight phase the aircraft is. For example, strong crosswinds are especially hazardous during take-off (FAA 2016a). A typical flight profile is made up of the taxing stage, take-off, departure, cruise, approach and land phases (Goblet *et al.* 2015). These phases are shown in Fig. 2.3 and are discussed below.

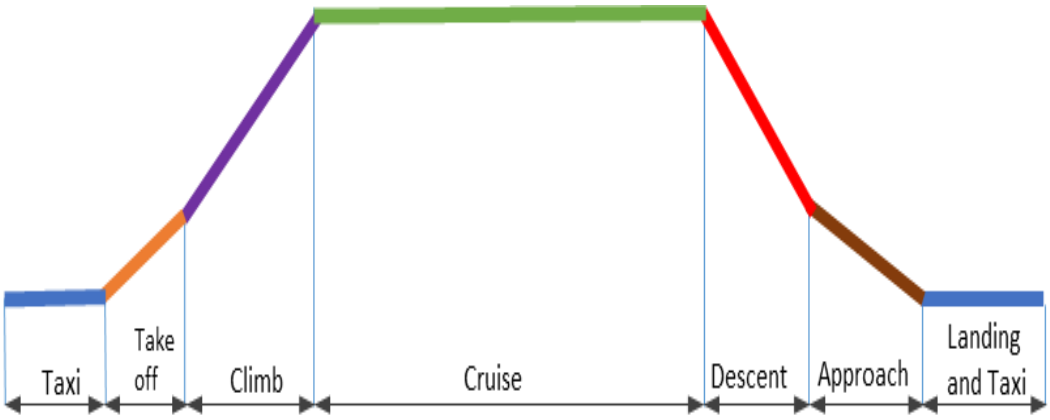


Fig. 2.3: Flight phases (FAA 2013a)

### 2.3.1 Take-off and Initial Climb

Take-off phase starts when take-off power is applied until the aircraft altitude reaches 35 feet above the runway. The phase also includes situations where the take-off process is started and aborted. Initial climb refers to the period between the end of take-off and the attainment of an altitude of 1000 feet above the runway or reaching the VFR pattern. The phase ends whenever any of these conditions is met.

### 2.3.2 En-route

En-route phase consists of climb to cruise, change of cruise level, descent or holding phases (CICTT 2013). Climb to cruise is the period from the end of initial climb to reaching the cruise phase. The cruise phase consists of level flight sections between reaching the initial cruise altitude and the beginning of descent to destination. Holding refers to carrying out a prescribed flight pattern while waiting for further ATC clearance. In VFR, the descent phase consists of decreasing altitude from the cruise phase to 1000 feet above the runway or to the

VFR pattern entry. For IFR flights, descent can be to the VFR pattern entry or to the Initial Approach Fix (IAF).

### 2.3.3 Approach

Approach is the phase of flight from 1000 feet above the destination runway or form entry to VFR pattern, to the landing flare (CICTT 2013). In the case of IFR flights, the approach phase starts from the IAF.

### 2.3.4 Landing

The landing phase starts from the landing fare until the aircraft stops on or exits the runway is exited (CICTT 2013). In touch-and-go landing, the phase ends at the application of take-off power. During the landing fare, the aircraft changes from a nose-low altitude to a nose-up position.

### 2.3.5 Taxing

Taxing could be before take-off (taxi in) or after landing (taxi out). During taxing, the aircraft moves under its own power on the airport surface (CICTT 2013). The phase also involves the power back sub-phase when the aircraft reverses from a parking position or a stand.

## 2.4 Weather data services and products

A major input to the weather avoidance process is weather (meteorological) data. Weather data is produced by numerous organisations and entities. Producing entities include national airspace agencies, aerodromes, and pilots. Aviation weather data sources and products include SIGMETs, AIRMETs, TAF, PIREPs and METAR (ICAO 2007a). These products are generated depending on specified conditions, as discussed in the following section.

### 2.4.1 SIGMET

Meteorological watch offices (MWO) issue SIGMET information whenever adverse weather phenomena occurs or is expected to occur en-route (ICAO 2007a). SIGMET messages also specify the geographical coverage of such phenomena that may impact aircraft safety. SIGMET messages are written in plain language using defined abbreviations.

### 2.4.2 METAR and SPECI

Aerodromes routinely release reports of weather observations, usually hourly (ICAO 2007a). Aerodrome reports contain indicators of the originating aerodrome, observation time, visibility, wind speed and direction, air temperature, altitude and other weather parameters. METARs are such kinds of report and are meant for flight planning and other uses that go beyond the originating aerodrome. SPECI reports are similar to METARS in that they are also meant for use beyond the originating aerodrome. However, SPECI cover special weather observations.

### 2.4.3 TAF

Terminal Aerodrome forecast (TAF) refer to forecast of the weather at an issuing aerodrome. TAF messages contain data such as indication of the originating aerodrome, time the TAF is valid, visibility and surface wind (ICAO 2007a).

### 2.4.4 AIREPS

Weather observations are made and reported by aircraft on international routes. Such reports are sent as soon as practicable via air-ground data link or voice communications (ICAO 2007a). Special aircraft reports are made whenever there is severe icing, severe turbulence, thunderstorms, volcanic ash cloud, or severe mountain waves. ATS units receive aircraft reports and forward them to MWOs and world area forecast centres (WAFCs).

### 2.4.5 AIRMETS

AIRMET messages are produced by MWOs and describe en-route weather that may impact the safety and operations of low-level aircraft (ICAO 2007a). Such information should not have been reported in the area forecast messages. The format of AIRMET messages is plain language text using abbreviations.

### 2.4.6 Weather radar

Ground-based weather radar provide display of precipitation or non-precipitation areas (FAA 2016b). The transmitted signals transmitted by the radar are reflected by targets and detected by the radar. The power of the radar return is generally dependent on the intensity and type of precipitation in the coverage area.

Weather radar could also be on-board, in which case areas of precipitation could be detected by the aircraft while in-flight. High reflectivity values (greater than 40dBZ) may indicate severe thunderstorms (FAA 2013b; Hupe *et al.* 2014).

### 2.4.7 Weather satellites

Satellites provide images of the earth that can be used for weather observations and forecast, Fig. 2.4. Satellite weather data are especially useful over remote areas, such as oceans, where other forms of weather data are insufficient or difficult to obtain (FAA 2016b). A notable satellite weather source is the Geostationary Operational Environmental Satellite (GOES) of the National Oceanic and Atmospheric Administration (NOAA).

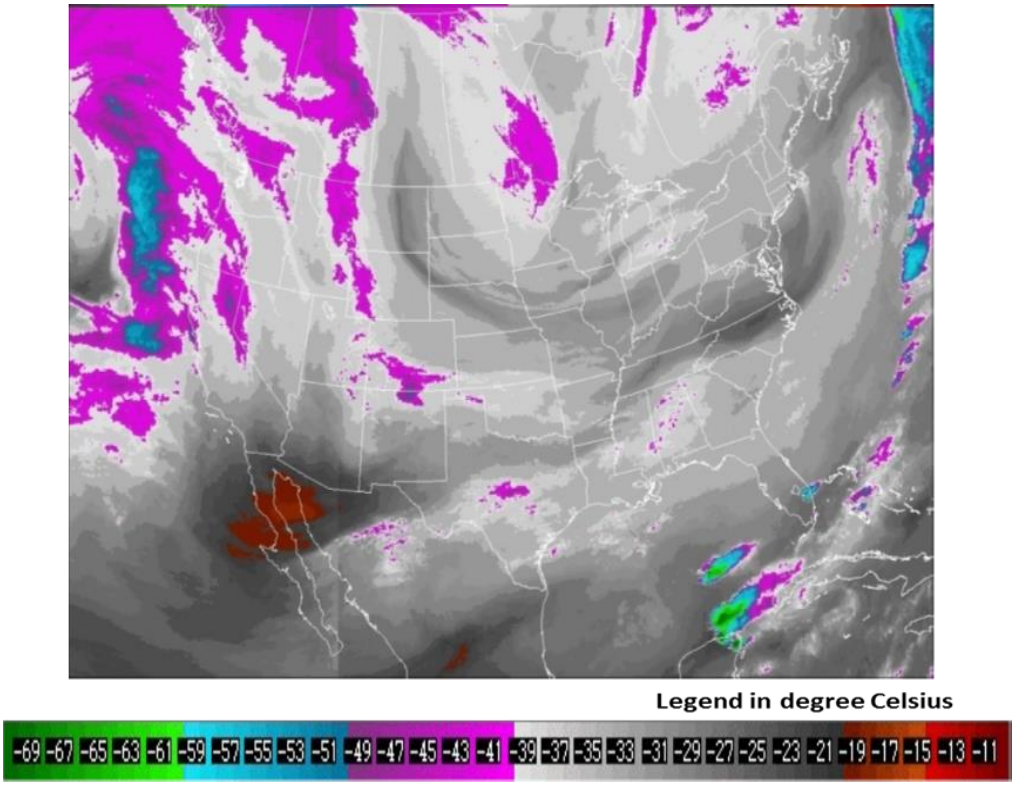


Fig. 2.4: Water vapour satellite imagery for troposphere layer between 10,000 ft mean sea level and flight level 390 (FAA 2016b)

Water vapour satellite imagery indicates the amount of water vapour in the atmosphere. Such imagery show the movement and location of thunderstorms and weather systems. Satellite visible imagery indicates the reflection of sunlight by the clouds, atmospheric particles and the earth's surface. Infra-red imagery indicate the amount of infra-red emitted by clouds, earth and particulate material.

The amount of emitted energy is generally dependent on the temperature of the targets.

The types of weather products and examples of weather characteristics they cover are summarised in Table 2.1 below.

Table 2.1: Summary of weather products for air transport (Data source: ICAO 2007a; FAA 2016b)

| **Weather Product** | **Class of weather** | **Weather phenomena and characteristics** | **Producers** |
|---|---|---|---|
| SIGMET | En-route adverse weather (Expected or Actual) | turbulence, thunderstorms | Meteorological watch offices (MWOs) |
| METAR and SPECI | Aerodrome weather observations | visibility, air temperature, wind speed and direction | Aerodromes |
| TAF | Forecast of weather at aerodrome | visibility and surface wind | Aerodromes |
| AIREPS | Weather observations by aircraft | severe icing, severe turbulence, thunderstorms and volcanic ash cloud | Aircraft, passed to meteorological watch offices |
| AIRMETS | En-route weather affecting low-level aircraft but not in area forecasts | thunderstorms, turbulence, severe icing | Meteorological watch offices |
| Weather radar | Display of precipitation areas | precipitation, severe thunderstorms | On-board aircraft tor ground-based |
| Weather satellites | Weather observations and forecast using earth images | water vapour, temperature, movement and location of thunderstorms and weather systems | GOES of National Oceanic and Atmospheric Administration (NOAA) etc |

## 2.5 Types of adverse weather

Weather hazards vary in types and impact. They also vary in terms of visibility. For example, thunderstorms are often visible, whereas clear air turbulence can be encountered without obvious warnings. Such weather hazards include

turbulence, thunderstorms, icing and fog. In the sections below, these weather phenomena have been discussed.

### 2.5.1 Turbulence

Turbulence refers to fast-changing variation in the speed and direction of air flow (FAA 2016a). It can cause an aircraft to move irregularly. Major types of turbulence include mechanical turbulence, convective turbulence and those caused by wind shear. The impact of turbulence depends on aircraft type and phase of flight (Sermi *et al.* 2015).

Mechanical turbulences occur when the smooth flow of wind is obstructed (FAA 2016a). This could be caused by objects such as mountains, trees and buildings. Such obstructions cause wind to flow in complex eddies. These eddies are further carried by the wind. Mountain waves constitute a type of serious mechanical turbulence. These types of turbulence develop downwind and on top of mountains. The force and speed of mountain waves could be high enough to drive aircraft onto the mountainside. The coverage of mountain waves may be up to 200,000 feet vertically and 1,000 kilometres from the mountain range (FAA 2016a).

Convective turbulence is caused by rising and downward convective currents (FAA 2016a). Warm air rises when the earth surface is heated, especially when winds are light. The convective current may rise to a level high enough for cumuliform clouds to form. Turbulence could be experienced in the resulting clouds, or beneath them. Thermals (dry convection) form when the air is very dry. In this case, aircraft may experience turbulence without the usual clouds that indicate such a possibility.

The rate of change in the speed or direction of wind with distance is known as wind shear (FAA 2016a). Wind shear can cause turbulence when two wind currents that have different directions meet. Turbulence can also occur when the currents have different speeds. A serious phenomenon associated with wind shear is clear air turbulence (CAT). It occurs in areas with no clouds and in high altitude (20,000 – 50,000 feet). The wind shear is often between a jet stream and surrounding air. CAT can impact aircraft unexpectedly.

### 2.5.2 Low Visibility

Clear vision is important for flight safety, especially for visual flying. However, many weather phenomena could obscure the vision of pilots, air traffic controllers and other stakeholders. Examples of such phenomena include dust storm, fog, smoke, volcanic ash and mist. In addition, visibility could be reduced by heavy snow or drizzle.

Dust storms occur when there are strong winds over areas with fine-grained soils (FAA 2016a). These areas include river flood plains, glacial deposits and dry lake beds. The lofted dust, in serious cases, can cause visibility to drop to nearly zero. In addition, dust can cause health problems and affect intake of air by aircraft engines. In these cases, flights become severely hazardous.

Fog occurs when air temperature is close to dew point (FAA 2016a). It is made up of tiny droplets of water that form a visible aggregate. Fog can reduce visibility to less than 1 kilometre. The base of a fog is on Earth's surface. Ice fog is a type of fog made up of ice crystals. Types of fog, based on cause, include advection fog, upslope fog, radiation fog, and steam fog. Advection fog occurs when moist air is cooled below its dewpoint when it flows over a colder surface. Similarly, upslope fog occurs when air moves up a slope. Steam fog is formed when vapour rises from a water body as cold air flows over it.

### 2.5.3 Adverse wind

Adverse wind refers to air flow that has speeds or changes of directions and duration that could negatively impact aircraft (FAA 2016a). Aircraft are most impacted by hazardous wind during take-off and landing. Adverse wind phenomena include variable wind, crosswind, wind shear, tailwind and gusts.

Variable winds change direction often. A wind flow that has variation in speed of more than 10m knots is known as a gust. A crosswind acts perpendicularly to aircraft heading. Crosswinds make it difficult to control the direction of aircraft movement and could lead to landing gear collapse. A tailwind produces a force behind an aircraft. This increases the roll required for take-off. When the speed of a gust suddenly increases, the lift on an aircraft can increase. Similarly, lift is suddenly decreased when gust speed drops.

### 2.5.4 Thunderstorm

Thunderstorm are storms that are marked by strong wind gusts, heavy rain, lightning and thunder (FAA 2016a). Turbulence, tornadoes, hail and icing can also occur in association with a thunderstorm. Because of their high vertical extent, thunderstorms can seriously impact air traffic. It is often difficult or not advisable for aircraft to fly through them, over them or under them.

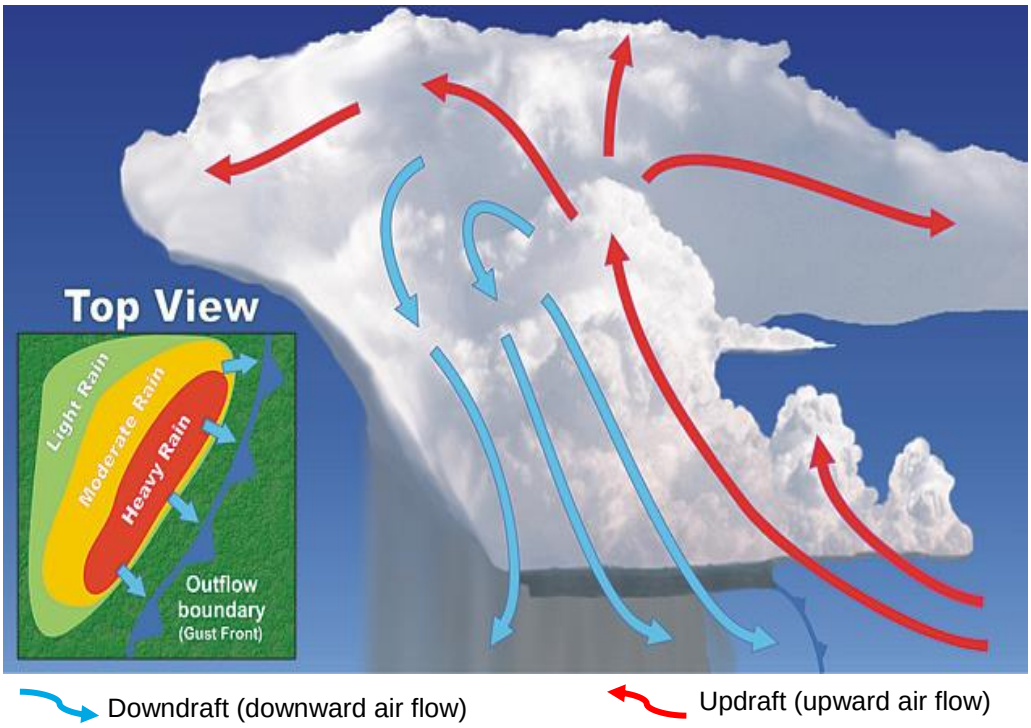


Fig. 2.5: Multicell thunderstorm (FAA 2016a)

When a thunderstorm is made up of one cell, it is known as a single cell thunderstorm. Thunderstorms can contain supercells or multicells. A supercell thunderstorm is mostly a single high-speed updraft that can exists for hours (FAA 2016a). Supercells are often associated with dangerous winds, large hail and other severe weather phenomena. Multicell thunderstorms are made up of combinations of single cells and supercells at various stages (Fig. 2.5).

### 2.5.5 Icing

Icing refers to the deposition of water in solid state (ice) on aircraft. Engine icing can affect aircraft engine performance by lowering incoming air temperature and flow (FAA 2016a). This leads to power loss. Structural icing, on the other hand, forms on the external parts of aircraft. Structural icing could be clear or rime. Clear

icing is formed when large water droplets slowly freeze. When the droplets are smaller and temperatures are colder, rime icing could be formed. Mixed icing occurs when both rime and clear icing form on an aircraft. In addition, severe icing can occur in mountainous areas. This is due to supercooled water droplets formed in upward wind flow in windward direction. Structural icing can adversely affect aircraft aerodynamics and airflow.

Weather phenomena and their impact are shown in Table 2.2 below. Further discussion of these effects of adverse weather also follows.

Table 2.2: Adverse weather phenomena and their impact on air transport

| **Weather** | **Definition** | **Impact** |
|---|---|---|
| Turbulence | fast-changing variation in speed and direction of air flow (FAA 2016a) | decreased passenger safety and comfort, accidents and decreased manoeuvrability of aircraft (EUROCONTROL 2013) |
| Low Visibility | lack of clear vision of airspace | accidents and decreased manoeuvrability of aircraft |
| Adverse wind | air flow with speeds or changes of directions and duration that could negatively impact aircraft (FAA 2016a) | accidents and decreased manoeuvrability of aircraft (FAA 2016a) |
| Thunderstorm | storms with strong wind gusts, heavy rain, lightning and thunder (FAA 2016a) | structural damage to aircraft, flight delay, increased pilot and ATC workload (Stewart *et al.* 2012; Hupe *et al.* 2014; FAA 2016a) |
| Icing | deposition of water in solid state (ice) on aircraft (FAA 2016a) | can affect engine performance and cause structural damage to aircraft (FAA 2016a) |

## 2.6 Impact of adverse weather

Adverse weather has considerable impact on flights and stakeholders. This includes structural damage to aircraft, loss of control, accidents, inconvenience to passengers, flight delays, increased fuel consumption and increased pilot workload.

### 2.6.1 Structural damage to aircraft

Adverse weather such as severe hail can cause damage to parts of the aircraft such as its airframe and engines (FAA 2016a). Increased water intake can occur in severe thunderstorms, leading to structural engine failure or flameout (FAA 2013b). This can result in controlled flight into terrain. In addition, the aerodynamic performance of the aircraft could be reduced, leading to more fuel being used than normal.

### 2.6.2 Accidents and decreased manoeuvrability of aircraft

Severe weather conditions can seriously impact the control of aircraft. This includes loss of lift, reduced directional control and low visibility (FAA 2016a). The impact is especially critical during take-off and landing phases. In some cases, accidents and loss of separation have resulted.

### 2.6.3 Passenger safety and comfort

Hazardous weather can affect passengers' experience of flight. For example, severe and unexpected turbulence make flights uncomfortable for passengers. In some scenarios, passengers can experience injury (EUROCONTROL 2013), especially when the adverse weather is unexpected.

### 2.6.4 Increased pilot and ATC workload

During adverse weather, pilots have more challenges and information than normal to process. In addition to operating and controlling the aircraft, they may have to plan, request and execute changes in the original flight path. ATC are also adversely affected. They ensure that airspace conditions, such as aircraft separation, are maintained. During adverse weather, they may receive more requests from pilots for clearance than usual. This leads to increased ATC communication workload (Stewart *et al.* 2012; Hupe *et al.* 2014).

### 2.6.5 Decreased airport and airspace capacity

The cautionary reduction in aircraft speed means less aircraft per time are able to use the weather-impacted runway and other airport facilities. During adverse weather, ATC often limit the number of aircraft per sector. These factors lead to decreased sector capacity (EUROCONTROL 2013) and may increase flight delays.

### 2.6.6 Increased fuel consumption

More fuel could be used because of issues such as use of longer alternative routes, reduced engine performance or impaired aerodynamic integrity (FAA 2016a). During adverse weather scenarios, aircraft may fly at non-optimal speeds or altitude. In addition, adverse weather mitigation procedures may involve air holding. These issues lead to inefficient use of time, fuel and crew resources.

### 2.6.7 Flight delay

Adverse weather and associated safety procedures is a major cause of flight delay (Hupe *et al.* 2014). During adverse weather pilots and ATC take extra caution and sometimes aircraft speed is reduced. This leads to flights arriving latter than expected. As a way of avoiding adverse weather, departure of aircraft are often delayed. In addition, weather avoidance manoeuvres may involve taking longer routes than usual, which could lead to longer flight durations.

## 2.7 Techniques for adverse weather avoidance

Exact methods, such as dynamic programming (Taylor and Wanke 2009), artificial potential field model (Xu *et al.* 2010) and Dijkstra's algorithm (Taylor and Wanke 2009), have been used for obtaining alternative routes that avoid adverse weather. However, these methods can take long times to produce solutions, if any. This is unacceptable for the short-term scenario considered by this work. Metaheuristic techniques provide suitable alternatives. Although the solutions they produce could be sub-optimal, they are generally faster and able to handle larger number of constraints. Such metaheuristic techniques include simulated annealing (SA) (Taylor and Wanke 2010), multi-objective genetic algorithm (GA)

(Alam *et al.* 2006; Kai-quan *et al.* 2015) and ant colony optimization (ACO) (Alam *et al.* 2006; Nguyen *et al.* 2007), and are discussed below.

### 2.7.1 Integer Programming

Integer programming (IP) optimisation (Roy and Tomlin 2007) was applied to a time-extended network model for aircraft routing. The model had the objective of minimising the sum of exit times *fx*, subject to flow constraints, sector capacity and forcing constraints (ensures certain paths are used for some flows), where $x_f$ is the flow on an edge (*v,w*). The advantage of this formulation is that it provides optimal solutions and there are well-known tools available. However, a major problem of the technique is the large computational time for non-trivial cases. The authors also suggested linear programming (LP) relaxation (Roy and Tomlin 2007). For that case, the values of $x_f$ did not have to be integers and were taken as real numbers instead. The LP technique was faster, but solutions could be fractional. For such scenarios, a first come first serve (FCFS) heuristic was used. The FCFS heuristic (Roy and Tomlin 2007) assigned each aircraft the first available slot and the aircraft is assumed to use the minimum possible time. Sector capacities are decreased with each aircraft passing through it. Fastest available route is assigned to the aircraft using breadth-first search (all nodes at a given depth is searched first before increasing the depth). In comparison with the IP and LP formulations, the FCFS heuristic technique had the least average running time, but its proposed solutions got worse with increasing traffic levels and adverse weather (Roy and Tomlin 2007).

### 2.7.2 Dijkstra’s algorithm

Dijkstra's algorithm was applied by Taylor and Wanke (2009) to generate flight reroutes during weather events. Fixes between the arrival and departure airports were modelled by network nodes. The graph edges represent connections between the fixes. The produced reroutes needed to be operationally feasible and flexible. Dijkstra's algorithm was used to generate *k*-shortest path. A weighted sum of acceptability metrics, such as reroute distance and lateral deviation, was used.

Given a weighted graph *G (V, E)*, the shortest path between any two nodes *P* and *Q* can be found using Dijkstra's algorithm (Dijkstra 1959; Fadzli *et al.* 2015). *E* is the set of edges (branches) and *V* is the set of nodes in the graph. The algorithm uses a bread-first search. For a given edge (branch) *e = (u, v)* between any two nodes *u* and *v* of the graph, a mapping *w(u, v)* specifies a positive weight for the edge. On the shortest path from *P* to *Q*, the sub-path from *P* to any of node $R_D$ is also the shortest path from *P* to $R_D$. The distance *d (v)* of a node *v* is the length of the shortest path from *s* to node *v*. The length of a path refers to the sum of the weights of its edges. Where the edge does not exist, the distance is ∞. The flowchart of the algorithm is shown in Fig. 2.6 (Qing *et al*. 2017).

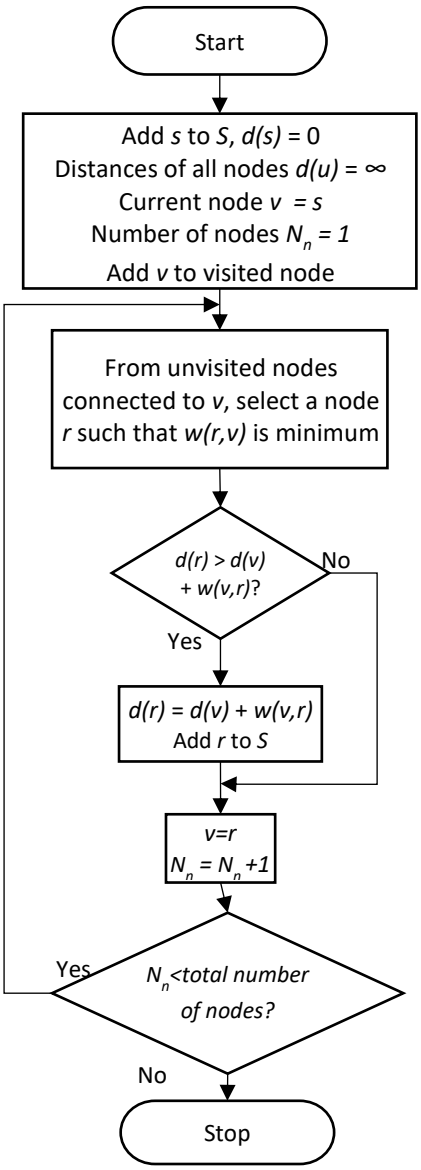


Fig. 2.6: Flowchart of Dijkstra's algorithm-based approach

For an initial node *s*, the algorithm proceeds as follows (Fadzli *et al.* 2015; Qing *et al*. 2017). Initially, all nodes have a distance of ∞, while *d (s)* set as zero. Node *s* is added to *S*, the set of nodes known to be part of the shortest path. The start node is taken to be the current node *v*. From the unvisited nodes connected to the node *v*, a node *r* is chosen such that edge *(r, v)* has the least cost. However, if the existing distance $d(r) > d(v) + w(v, r)$, *r* is added to *S*. In addition, $d(r)$ is given a value equal to $d(v) + w(v, r)$. The process is repeated until the destination node is reached.

### 2.7.3 A* algorithm

To increase the speed of Dijkstra's algorithm, the A* algorithm has been proposed (Xie and Zhong 2016). The A* algorithm differs from Dijkstra's algorithm in that a heuristic function is added to the cost (weight) function $C(n)$, i.e.

$$C(n) = g(n) + h(n) \tag{2.2}$$

where $g(n)$ and $h(n)$ are the distances from a current node *n* to start node and destination nodes respectively.

An example of a heuristic function could be the distance from a node to the destination node. The performance of A* algorithm is dependent on choosing a suitable heuristic.

The A* algorithm has been applied by Xie and Zhong (2016) to the problem of finding efficient flight paths during adverse weather. A two-dimensional grid system was used for adverse weather avoidance modelling. However, the work did not provide adequate evaluation of the impact of adverse weather on flights. Moreover, the performance of the A* algorithm is often inadequate in complex environments (Muñoz *et al.* 2014), such as those found in operational flight routing.

### 2.7.4 Simulated Annealing (SA)

SA (Taylor and Wanke 2010; Chaimatanan *et al*. 2012) is a heuristics inspired by the cooling process used to produce metals with desired properties. An initial flight solution *x* with cost $J(x)$ is perturbed, by randomly adding, removing, replacing fixes, or increasing/decreasing flight delays (Taylor and Wanke 2010). Perturbed solutions must satisfy operational constraints (remove routes with

cycles/repeated nodes and large turn angles). The change in operational costs after perturbation is calculated. If it is negative (meaning lower costs), the new solution $x_p$ is accepted. If the change in cost $(\Delta J)$ is not lowered, $x_p$ may still be accepted, depending on the probability (Taylor and Wanke 2010):

$$P(\Delta C) = \exp(-\Delta J / T) \quad (2.3)$$

Temperature $T_k$ at the $k$th iteration is decreased using an exponential cooling schedule, defined as (Taylor and Wanke 2010):

$$T_k = \Delta T^k * T_o \quad (2.4)$$

where $T_o$ is the user-defined initial temperature. The process is stopped once the system is frozen or a user-defined criterion is reached.

### 2.7.5 Ant colony Optimization (ACO)

The ant colony system algorithm (ACS) (Dorigo and Gambardella 1997) mimics the path-finding behaviour of ants attempting to reach a food source. For each path they use, the ants lay chemicals called pheromones. The most-used path has the strongest pheromone concentration. This path is usually the shortest path to the food source, since ants that use it are able to return more often to the food source.

In applying ACS to a graph of nodes connected with edges, $m$ ants are placed on $n$ nodes. The initial placement of the ants could be done randomly, for instance. A tour is constructed by each ant using the state transition rule, which greedily determines the next node $s$ to which the ant $k$ on node $r$ would move to (Dorigo and Gambardella 1997). The rule provides the balance between exploration and exploitation, depending on a uniform probability $q$.

Exploitation means that the node to which the ant moves to next is the one with the highest combination of the inverse of link length and pheromone concentration $\tau$. The parameter $\beta$ determines the relative importance that is given to pheromone information.

Exploration is carried out by choosing the next node randomly. The ant uses the local update rule to modify the amount of pheromone on the visited edge to a weighted sum of the previous concentration and a pre-defined increment. For

each iteration, after all ants have completed their tours, the ant with the best tour updates the pheromone amount on the edges it visited. The pheromone values are updated using the global update rule (Dorigo and Gambardella 1997):

$$\tau(r,s) = (1-\alpha).\tau(r,s) + \alpha.\Delta\tau(r,s) \quad (2.5)$$

where $\alpha$ is the pheromone decay parameter, $\Delta\tau(r,s) = 1/L_{gb}$ if link $(r,s)$ is part of the best global tour and $L_{gb}$ is the length of the best tour. The global update rule gives greater amount of pheromone to edges belonging to the globally best tour.

Preliminary work on the use of ACO to obtain weather-avoiding paths in a 4D grid has been described in (Nguyen *et al.* 2007). The time dimension was considered for a 3D grid by taking snapshots of the aircraft and weather cells positions at regular intervals. Each grid cell stores array elements that contain information about feasible next links and possible manoeuvres to reach them. The array element also stores the snapshot index and position.

The objectives of their approach were to minimise heading/altitude changes, minimise deviation from flight plan and avoid weather cells. In applying ACO to the adverse weather avoidance, Nguyen *et al.* (2007) defined the visibility parameter $\eta(r,s)$ as:

$$\eta(r,s) = 1.0/(w_1WF + w_2HF + w_3AF + w_4DF) \quad (2.6)$$

where $w_1 \ldots w_4$ are weight factors and are used to determine how much each of the factors (*WF*, *HF*, *AF* and *DF*) contribute to the total cost, *WF* = weather severity factor, *HF* = heading change factor, *AF* = altitude change factor and *DF* = estimated distance from node *s* to exit point.

More work needs to be done to compare the performance of the ACO algorithm with alternatives for the 4D scenario. In addition, its performance in the presence of large number of weather cells and high traffic levels would be of interest.

### 2.7.6 Artificial Potential Field Model

Xu *et al.* (2010) proposed an improved artificial potential field technique for rerouting of aircraft paths during severe weather. In this method, an aircraft was modelled as subject to repulsive forces due to obstacles, and an attractive force due to the destination. The resultant force determined the actual speed and

direction of the object. To solve the local minima issue in which the aircraft was stuck before obstacles collinear to the destination, an additional gravitational point was defined. The effect of this was to reduce the repulsive force field as the destination was approached. A repulsive field function with faster rate of change was also defined. This was to make the aircraft move faster away from obstacles. The advantage of their method was that it considered aircraft cruise phase, was faster than the traditional artificial field model and avoids the local optimal of collinear adverse weather and destination. However, the defined model did not adequately consider multiple flight link costs.

### 2.7.7 Genetic algorithm

Genetic algorithms have been suggested for obtaining conflict-free flight trajectories (Alam *et al.* 2006; Kai-quan *et al.* 2015). Genetic algorithms mimic the natural process of reproduction. Candidate solutions are initially (often randomly) generated and represented by structures known as chromosomes. The non-dominated sorting genetic algorithm II (NSGA-II) (Deb *et al.* 2002) is an elitist multi-objective genetic algorithm that sorts solutions of a randomly-generated parent population according to their non-domination level. An offspring population is created from the parent population using binary tournament selection, recombination (crossover) and mutation. The parent and offspring population sets are combined and sorted. The best members of the resulting set form the parent population for the next generation. The process is repeated until the termination criterion is reached.

Kai-quan *et al.* (2015) used NSGA-II to solve the network-wide conflict-free flight trajectories planning problem. They attempted to minimise ground holding, airborne delay and flight level allocation costs. Each chromosome of candidate solutions had genes representing the waypoints, flight level and timeslot allocation of the flight. To accelerate the algorithm, they defined a (greedy) heuristic to allocate alternate routes whenever a conflict is detected. However, they did not consider the effect of weather avoidance in their work.

Lee *et al.* (Ping 2003; Lee *et al.* 2007) proposed a GA model that improves the robustness of airline scheduling by adjusting flight departure time. They developed a model which determines the shift in time $\Delta_i$ that needs to be applied

to each flight leg *i* with departure time $x_i$ to minimise costs. Whereas (Lee *et al.* 2007) focussed on pre-departure flight improvements, this work considered aircraft in flight. The present work used a graph-based model to find optimal routes using multi-objective shortest path; their work used a scheduling model. While the present work also used multi-objective genetic algorithm, it used the less computationally intensive weighted sum approach to calculate fitness function. The faster computation was useful for the enroute scenario considered in this work.

### 2.7.8 Comparison of the weather avoidance methods

Table 2.3 shows the comparison of the various techniques for adverse weather and aircraft avoidance.

Table 2.3: Comparison of techniques for weather and aircraft avoidance

| **Technique** | **Type** | **Model** | **Considered aircraft avoidance** | **Advantages** | **Disadvantages** |
|---|---|---|---|---|---|
| Integer programming (Roy and Tomlin 2007) | Deterministic | Network graph | No (Sector capacity considered) | Can obtain the best path | - Does not scale well |
| Dijkstra's algorithm (Taylor and Wanke 2009) | Deterministic | Network graph | No | Can obtain the least cost path | - Does not scale well |
| A* algorithm (Xie and Zhong 2016; Fadzli et al. 2015) | Heuristic | Network graph | Yes (horizontal plane) | Can be faster than Dijkstra's algorithm | - Trapped in local optima (Xie and Zhong 2016)<br>- Need to find suitable heuristic function |
| Simulated Annealing (SA) Taylor and Wanke (2010) | Heuristic | Network graph | No | Lower computational time and effort | - Might get trapped in local optima (Chaimatanan et al. 2012) |

Table 2.3: Comparison of techniques for weather and aircraft avoidance (cont'd)

| **Technique** | **Type** | **Model** | **Considered aircraft avoidance** | **Advantages** | **Disadvantages** |
|---|---|---|---|---|---|
| Ant colony Optimization (ACO) Nguyen *et al.* (2007) | Heuristic | Grid | Yes | Good exploratory search (diversity) | - Can be trapped in local optima (Wang and Yang 2013) |
| Artificial Potential Field Model Xu *et al.* (2010) | Deterministic | No | No | - Supports moving weather<br>- Fast computation | -Trapped around obstacles and in local optima |
| Genetic algorithm (Kai-quan *et al.* 2015; Lee *et al.* 2007) | Heuristic | Network graph | Yes | Can find global optima (Xie and Zhong 2016) | - Did not consider weather regions<br>- Sometimes trapped in local optima |

The considered techniques could be exact (deterministic) in the sense that they are guaranteed to produce the same results each time they are run with the same inputs. The result obtained is usually the best solution. This contrasts with heuristics which produce good but often sub-optimal solutions. However, heuristics often require less computational effort or time.

Another basis for comparison between the techniques is the way the airspace is modelled. These modelling methods include network graphs, where the airspace is represented by waypoints and links between them. In addition, the airspace could be modelled by grids or discrete steps through the continuous airspace. Some of the techniques considered the avoidance of other aircraft in the airspace. In addition, the table looked at the advantages and disadvantages of each technique.

Integer programming (Roy and Tomlin 2007) is an exact technique that can obtain the best possible path. However, the disadvantage of the approach is that it does not scale for scenarios with large number of waypoints and constraints. Whereas the approach of Roy and Tomlin (2007) supported the inclusion of ATC sector capacity, the avoidance of other aircraft in the airspace was not considered.

The Dijkstra's algorithm approach (Taylor and Wanke 2009) for weather avoidance used a network graph for airspace modelling. The advantage of Dijkstra's algorithm is that it is exact and can produce the shortest path, especially for simple cases. The disadvantage of the approach is that it did not directly consider avoidance of other aircraft in the airspace, or similarly complex airspace scenarios.

To improve the speed of Dijkstra's algorithm, the A* approach (Xie and Zhong 2016) used a heuristic in obtaining the alternative paths for adverse weather scenarios. The A* algorithm can often obtain the best path faster than Dijkstra's algorithm. However, there is the challenge of deriving a heuristic with suitable performance. In addition, the A* algorithm can sometimes get trapped in local optima (Xie and Zhong 2016).

The simulated annealing approach of Taylor and Wanke (2010) used a network graph for airspace modelling. The work did not consider the avoidance of other aircraft. The simulated annealing technique is a heuristic and has the advantage of lower computational time and effort. However, the approach has the problem of getting trapped in local optima (Chaimatanan et al. 2012).

The ACO-based approach of Nguyen et al. (2007) for weather avoidance used a hyperrectangular grid model for airspace representation. The ACO technique had the advantage of good exploratory ability. This ensures that there is diversity in candidate solutions. However, the heuristic can get trapped in local optima. The work supported avoidance of other aircraft.

The improved artificial field model technique used by Xu *et al.* (2010) is an exact method but can be trapped around obstacles and local optima. The approach used step movements to search a continuous airspace model when deriving alternative paths. The advantages of the approach are its fast computational speed and support for moving weather. The work did not consider the avoidance of other aircraft.

Genetic algorithms (Cai *et al.* 2015) have been used for flight rerouting with a graph airspace model. The avoidance of other aircraft was supported by their work. Genetic algorithms have the advantage of finding the global optima. However, the algorithms can still get trapped in local optima in complex problems.

## 2.8 Related work in weather avoidance

This section reviews a number of related work in the area of flight path routing during adverse weather avoidance. The studies covered the avoidance process during various phases of flight. In addition, the various methods of airspace modelling for weather avoidance has been discussed.

### 2.8.1 Departure and pre-departure weather avoidance

Adverse weather avoidance process could be done before departure. This ensures that the aircraft avoids any phenomena that was known before departure. However, this approach does not adequately consider unexpected weather that occurs while the aircraft is in flight. In addition, some research have considered the avoidance of adverse weather during departure.

An example study for weather avoidance during departure is ACROSS (Advanced Cockpit for Reduction of Stress and Workload). The study considered the development of a decision support tool to enable aircraft avoid hazardous terrain and weather during departure (Cauchi *et al.* 2015). Such hazardous weather could significantly impact aircraft safety and increase the workload of flight crew. The work intended to address the problem that the avoidance process was manual. In addition, crew needed to acquire data from multiple sources: ATC, weather forecast and on-board weather. They proposed a partially-automatic tool (algorithm and HMI) to assist pilots. Fig. 2.6 shows the diagram of its functional components. The system detected weather conflicts along SID (Standard Instrument Departure). Various options were presented by the tool such as delaying take-off or flying alternative paths to avoid hazardous weather regions. In addition, the tool ensured that safe separation from terrain was maintained. After selecting the appropriate route, pilots then requested clearance from ATC for the route.

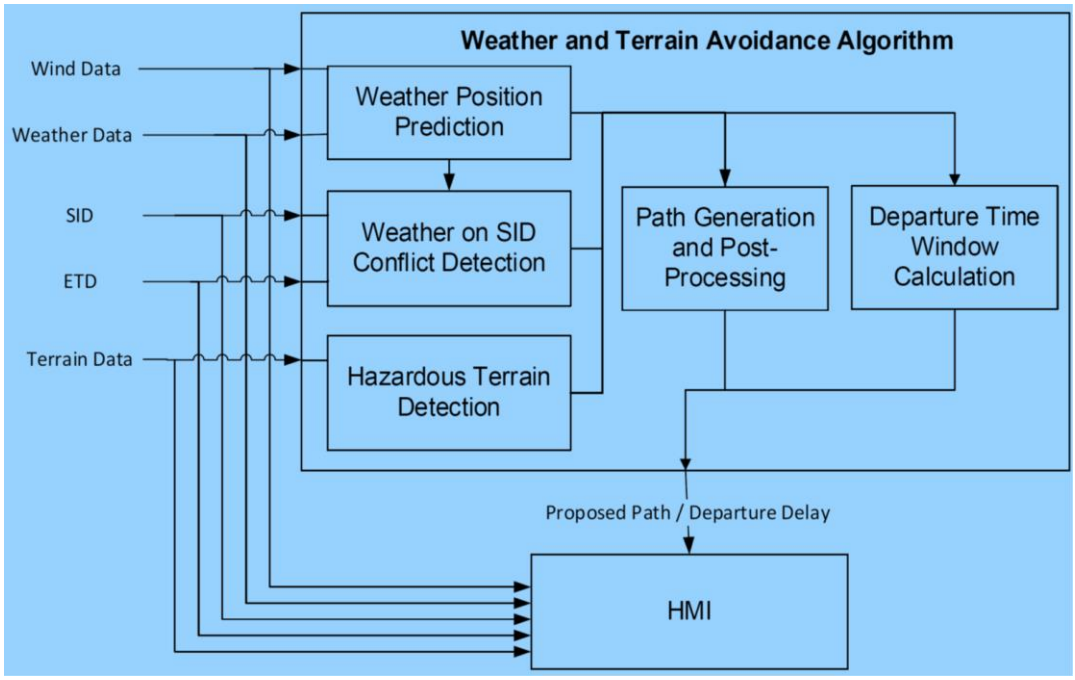


Fig. 2.7: The ACROSS weather avoidance algorithm (Cauchi et al. 2015)

An A* algorithm was used for routing. Inputs to the tool included an obstacle map model, represented by a 2D grid. Weather cells were represented using 2D polygons. Each polygon had different velocity. The aircraft's turn radius was used by the system for path smoothing. The paper, however, did not adequately consider the impact of the process on passengers.

Ng *et al.* (2009) developed a system that produced flight paths during convective weather such that system efficiency and throughput were optimised (Ng *et al.* 2009). The algorithm used was linear with number of links and is based on dynamic programming. The route deviation and fuel cost were the minimised costs. To reduce computational time, the search space is grouped into grids, which are then analysed. Generated routes were 20 nmi shorter than Coded Departure routes (CDRs), defined by the FAA for use before departure during bad weather or traffic congestion scenarios.

Chaimatanan *et al.* (2014) attempted to develop more efficient strategic trajectory planning (one day in advance) required to handle increasing global air traffic. They also addressed the problem of minimising global interaction between 4D aircraft trajectories at continental scale (Europe). This was done by adjusting departure times and trajectories. Such interactions included lack of time

separation, violation of minimum (vertical and horizontal) separation, distance between trajectories and topology of trajectory intersection.

A mixed-integer optimisation problem was used to model the route and departure time assignment (Chaimatanan *et al.* 2014). To consider uncertainty in aircraft positions, the protection radius around each aircraft is enlarged by a defined uncertainty margin. Each initial route is assumed to be optimal for the airliner. The applied routing technique was a hybrid-metaheuristic optimisation algorithm using simulated annealing, with hill climbing algorithm as local search technique when a defined probability was met. The work also used a hash-table method to detect interactions between trajectories. In the hash-table method, each trajectory point was mapped to a grid cell to determine if the minimum separation requirement was violated. If the cell was occupied by another aircraft, then a conflict was noted. The developed system did not adequately consider the impact of adverse weather and other scenarios where there are fast and substantial changes.

Route planning with turn constraints under hazardous weather scenarios was presented by Krozel *et al.* (2006). Routes were generated between given start points and destinations, such as from sector boundary crossing to airport metering fixes. The second example was routes from airport metering location to runway final approach fixes. Constraints include avoiding special use airspace and considering aircraft dynamics such as velocity and acceleration. Arrival and departure corridors were also considered, as well as pilot and controller workload.

A dynamic programming algorithm was used to search in a discretised geometric model of an airspace (Krozel *et al.* 2006). The model was 2D, with possible extension to 3D. The algorithm searched for optimal paths with given range of number of turns. The method was based on Bellman-Ford shortest path algorithm and post-processing was carried out for local route optimisation. Data input was Vertical Integrated Liquid (VIL) data of the Integrated Terminal Weather System (ITWS). The developed system was applied to a scenario using Dallas Fort Worth Airport (DFW). Results indicated that weighted route length was minimised, given turn constraints. In addition, routes were produced that avoided adverse weather around DFW. For randomly generated pairs of location and headings along the

metering fix, the algorithm generated feasible route with at most *k* links. However, surrounding air traffic information was not considered adequately.

Zhang *et al.* (2012) proposed a decentralised framework for obtaining 4D flight plans for large number of aircraft. The paper considered the weather and capacity constraints of the overall system. The airway network was defined by a directed graph model. The waypoints in the airspace were represented by nodes in the graph and the links of the graph represented high-altitude jet routes (directed airway). The network had non-overlapping subsets (sectors) with associated maximum capacities. The problem was modelled by a time-dependent shortest path and solved using dynamic programming. Users could obtain trade-offs between departure delay, expected turbulence, total travelling time, fuel consumption and so on. Flight paths were obtained before departure.

Two severe convective weather storms were considered (one totally covering some sectors, and another partially covering other sectors causing their reduced capacity). Results indicated that the algorithm was able to produce paths that met capacity requirements and had more uniform use of airspace sectors. For considered adverse weather scenario, flights were able to leave on time, while incurring 0.52% increase in flight distance. However, actual performance for large-scale scenarios was not adequately addressed.

### 2.8.2 Weather avoidance during arrival

Adverse weather reroutes could also occur during approach to a destination airport. The Dynamic Arrival Routes (DAR) research developed a system for avoiding adverse weather during flight arrivals (Gong *et al.* 2015). The phenomenon studied was convective weather. Airborne flights were analysed continuously to determine routes around adverse weather. The developed system was integrated with arrival scheduling. The results showed average savings of 12.3 minutes for look ahead times of 60 minutes from metering fix. In addition, weather avoidance reroutes were free of adverse weather 83% of the time for 30 minutes look-ahead time from meter fix. Trajectory updates were at the radar update rate of 12s.

### 2.8.3 En-route and tactical weather avoidance

Some of the related research considered en-route phases of flight, such as the cruise phase. Windhorst *et al.* (2009) considered the avoidance of convective weather when the forecasts have uncertainty. The work mostly considered the reroute of aircraft in cruise stage. Flight trajectories were generated using a 4-DOF flight model with parameters obtained from Base of aircraft Data (BADA). The trajectories had resolutions of one minute. A resolver algorithm determines reroutes by generating auxiliary waypoints around weather polygons. However, the algorithm was not optimal for multiple weather polygons located downstream of a route. Uncertainty in weather predictions was considered by periodically generating new routes based on weather updates. The phenomena of interest was convective weather, including thunderstorms. System inputs included CWAM forecast. From reported results, the system was able to resolve 79% of detected encounters, given a planning horizon of 20-120 minutes and interval of 15 minutes. Average delay incurred due to reroutes was 3.3%.

Dynamic Weather Routes (DWR) was a system that produced adjustments to near-time trajectories of en-route aircraft (McNally *et al.* 2012). This was to ensure time and fuel savings. The automated and ground-based system suggested more efficient flight reroutes to air traffic managers and dispatchers during convective weather. Inputs to the system included archived traffic and weather data from FAA Fort Worth Air Route Traffic Control Centre (ARTCC) and national Airline Situation Display to Industry (ASDI) data. Aircraft that were in-flight are continuously analysed, in order to obtain alternative routes.

To determine the paths of flight and moving weather cells, the Corridor Integrated Weather System and Convective Weather Avoidance Model was coupled with the Centre/TRACON Automation System (CTAS). The routing algorithm and 4D trajectory predictions were provided by the auto-router and CTAS components respectively. The suggested routes were tested for weather and traffic conflicts. In addition, wind-corrected flight delay and time savings were evaluated.

Results showed that about 10 minutes of flying time could be saved, on the average. Most flight paths proposed by DWR had lower track miles, compared with those of historical reroutes. In addition, trajectories obtained by DWR led to

up to 55% reduction in sector congestion times. For the considered scenario, the highest frequency of DWR request was two per sector per time slots of 15 minutes. This did not appear overwhelming. One drawback of the study was that performance of the system on-board aircraft was not evaluated. In the on-board scenario, the continuous analysis function may incur extra data costs and require continuous connection to multiple data sources.

Hupe *et al*. (2014) considered adverse weather avoidance modelling using the MET4ATM system. The work presented the application of a weather avoidance model (DIVMET) connected to an air traffic model (NAVSIM). DIVMET was used to generate trajectories based on weather radar polygons. Given an affected region divided into left and right areas by the original path, the DIVMET model selected reroutes on the side with a smaller area (Hauf *et al.* 2013). The intention was to find what side of a weather polygon (left or right) produced routes with the smaller length. However, the method was not designed to select the most efficient path around weather-affected areas. The weather phenomenon considered was thunderstorm. Input to the system included radar data and the values for the safety margin from weather phenomena. The avoidance options included the ability to delay flight departure or to return to the original planned route at predefined waypoints. Results showed that larger safety margins led to higher levels of deviation of simulated routes from the middle of original flight route. In addition, simulated routes obtained by the developed system were shorter than routes previously flown by affected aircraft.

The goal of the work of Lim and Zhong (2017) was to produce routes that have minimum changes to flight distance while avoiding regions with dynamic convective weather and prohibited airspace regions. Areas where flights are not allowed at all times are known as prohibited areas, whereas flights are allowed in restricted areas when certain conditions are fulfilled. At certain times, danger areas have activities that are risky to aircraft. The paper used grid-based cellular automaton for optimisation of the air route network. However, the work did not consider vertical movement of aircraft and only cruising phase was considered. The application of the system was demonstrated with a use case of Myanmar airspace. For the case studies considered, the obtained flight routes were able to

avoid the three areas and dynamic convective weather cells, with a maximum increase in flight distance of 39.53%.

Autonomous Operations Planner (AOP) included a set of algorithms for strategic and tactical conflict detection and resolution (Karr *et al.* 2012). The system also ensured that hazard avoidance and route constraints are satisfied. The considered hazards included convective weather and restricted airspace, whereas route constraints included times of arrival. The tool was designed for supporting flight crew in self-separation or in obtaining routes more likely to receive clearance from ATC. Self-separation was suggested as a way to solve the challenges of increased ATC workloads, traffic congestion and unpredictable local weather. The project indicated that automation of self-separation was important because of the difficulty for pilots to grasp all interactions between aircraft applicable to them. In addition, pilot manoeuvres to avoid one hazard may lead to the encounter of another hazard. Modified (MOD) routes could be received by data link, created by manual input from flight crew or obtained using AOP. In addition, the system had the capability to detect conflicts resulting from proposed changes to existing settings. The platform modelled 4D trajectory prediction within uncertainty bounds.

AOP supported both strategic and tactical flight mode. In strategic flight mode, lateral navigation (LNAV) and vertical navigation (VNAV) of the flight management system (FMS) were activated for the current route (Karr *et al.* 2012). Tactical flight modes were other modes that are not strategic. Intent-based function referred to those functions that considered the aircraft intent (final goal). State-based functions were tactical and did not consider aircraft intent. State-based conflict resolution used the aircraft's position and velocity, as well as those of surrounding air traffic. It was used as a recovery fall-back whenever intent-based conflict resolution failed. For conflict resolution in strategic intent-based mode, a pattern-based genetic algorithm (PBGA) was used. In the PBGA technique, a manoeuvre pattern created a manoeuvre away from a given route at one point and caused a return to it at another point. Not much quantitative evaluation was presented by the paper.

Sauer *et al.* (2017b) considered the application of DIVMET to flight execution in scenarios with multiple hazards due to weather. A major trade-off was between

extra flight distance and weather encounters. Two weather situations were considered: winter frontal system and air mass convection, including turbulence and icing. Weather data input was from Consolidated Storm Prediction for Aviation (CoSPA), Current and Forecast Icing Product (CIP/FIP), Graphical Turbulence Guidance Nowcast (GTGN). Various combinations of weather severity were considered, for example, severe and moderate weather hazards. Rerouting was lateral and two-dimensional. Results for a considered scenario with 290 flights showed that severe convection avoidance caused an increase of 5 nmi on the average and 156 nmi maximum. When moderate hazards such as turbulence were included, long detours were obtained, with average increase in distance between 98 and 184 nmi.

Sauer *et al.* (2017a) investigated the impact of multiple hazards on aircraft routing and flight. Simulations using DIVMET were presented. The scenarios involved early winter frontal system and convective weather with isolated cellular convection. Both cases had areas of turbulence, icing and convective storms. The research was similar to Sauer *et al.* (2017b), but considered effects of wind and controller workload. For 1740 routes considered in the early winter scenario, 16.4% of obtained routes encountered fewer ATC sector than previously planned. 13% of the routes entered up to 4 sectors. For the summer scenario, 11% of the routes had reduced number of ATC sectors than previously planned. 7.5% of the flights entered two more sectors than planned. When the impact of wind was considered, additional flight times were up 115 minutes in the November 2015 scenario. In the summer scenario, it was 71 minutes. Favourable winds for maximum of 3 flights reduced flight time by 2 minutes. Deviations in the November 2015 scenario (effect of wind not considered) were up to 86% of distance of the original route, while in the summer case (effect of wind not considered), route distance had a maximum increase as high as 40%.

### 2.8.4 Airspace modelling approach for weather avoidance

Different modelling approaches could be used for the determination of the best path reroutes. These approaches for modelling airspace include network graph methods, flow-based techniques or grid models.

Wang and Yang (2013) presented an ant colony algorithm for rerouting. The work used a grid-based method. The flight path from Guiyang-Changsha was studied. The weather of interest was thunderstorm phenomenon. Weather data was obtained from meteorological satellites and Doppler weather radar. Simulation results showed that there was an improved utilisation of airspace. Local optima were also avoided for the scenario studied. However, only sideways (lateral) avoidance in a 2D environment was covered in the work.

The En route Flow Planning Tool (EFPT) was designed for tactical management of traffic flow by air traffic managers (Stewart *et al.* 2012). The developed system supported both automated and manual generation of reroutes. The tool was flow-based and used a more accurate Corridor Integrated Weather System (CIWS) forecast. CIWS Vertically Integrated Liquid and echo tops were used as inputs to the system. An algorithm was defined for weather impact identification. The system used a weight sum of metrics such as sector congestion, weather avoidance, flow agreement (gives more values to route segments that have been more commonly flown) and flight distance.

Taylor *et al.* (2017a) considered the use of network optimisation in the design of flight reroutes. A set of reroutes were generated and stakeholders could choose an appropriate route from the set. The considered network model included segments that had been previously flown by the aircraft. An improved Dijkstra's shortest path (DSP-M) algorithm was used to obtain routes that are constrained to pass through defined intermediate nodes ($m$). Metrics were defined, such as flow conformance and relative schedule disruption that depends on departure time. The research supported direct calculation of multiple metrics using the weighted sum of the metrics or the use of Pareto solution set. The reroutes were more likely to be operationally acceptable, although they may not be the shortest routes. The paper considered examples of reroutes around convective weather in the United States.

In a related work, a platform for obtaining reroutes during tactical avoidance was presented (Taylor *et al.* 2017b). Candidate set of reroutes generated using the modified Dijkstra's shortest (DSP-M) algorithm. A multi-objective genetic algorithm (MOGA) evaluated trade-offs between multiple metrics, so as to avoid problem of defining weights for a weighted sum approach. Thereafter, distinct

groups of routes were identified using principal component analysis. This was to maintain diversity while reducing the number of advisory sets and dimensionality of the problem. Clusters of solutions that have specified trade-offs were obtained using spectral clustering algorithm. Solutions that represent each cluster were then selected.

Table 2.4 shows the comparison of the various related research, in terms of characteristics such as avoidance technique, considered weather and dimensions.

Table 2.4: Comparison of adverse weather avoidance systems

| **Project/ Author** | **Weather phenomenon** | **Avoidance Technique** | **Dimension of Avoidance** | **Flight Phase** | **Aircraft avoid-ance** |
|---|---|---|---|---|---|
| ACROSS (Cauchi *et al.* 2015) | Precipitation (weather cells) | A* | 2D | Departure (before take-off) | Partial, requested ATC clearance |
| Wang and Yang (2013) | Convective weather, Thunderstorm | Modified ant colony algorithm | 2D | Cruise | No |
| MET4ATM | Thunderstorm | DIVMET geometric heuristic (Hauf *et al.* 2013) | 2D | Generic (included en-route) | Yes |
| DWR (McNAlly *et al.*) | Convective weather | Autoresolver using tangent rays (Erzberger *et al.* 2010) | 4D | En-route | Yes |
| Krozel *et al.* (2006) | Precipitation weather cells/ storms | Dynamic programming | 2D | From metering fixes to final approach fixes (b) sector crossings to metering fix destination | Partial: considered ATC |
| Hierarchical flight planning | Convective weather storms | Dynamic programming | 4D | Before departure | Partial: considered ATC |
| Chaimatanan *et al.* (2014) | - | Simulated annealing & hill climbing algorithm | 4D | Strategic/ en route (before departure) | Yes |
| Dynamic Arrival Routes (DAR) (Gong *et al.* 2015) | Convective weather | Autoresolver (like DWR) | 2D | Arrival | Partial |

Table 2.4: Comparison of adverse weather avoidance systems (cont'd)

| **Project/ Author** | **Weather phenomenon** | **Avoidance Technique** | **Dimension of Avoidance** | **Flight Phase** | **Aircraft avoid-ance** |
|---|---|---|---|---|---|
| Dynamic programming flight routing algorithm (Ng *et al.* 2009) | Convective weather | Dynamic programming | 3D | Pre-departure | No |
| En route Flow Planning Tool (EFPT) (Stewart *et al.* 2012) | Convective weather | Custom heuristic | - | Tactical airborne | Partial: depended on ATC |
| Windhorst *et al.* (2009) | Convective weather, thunderstorms | Geometric algorithm | 2D | Cruise | Yes |
| Autonomous Operations Planner (AOP) (Karr *et al.* 2012) | Convective weather | Genetic algorithm (PBGA) | 4D | Airborne | Yes |
| Lim and Zhong (2017) | Convective weather | Cellular automaton | 2D | Cruising | No |
| Node-based trajectory prediction and turbulence avoidance (Cheung 2017) | Clear Air turbulence | A* | 2D | Before take-off | No |
| Taylor *et al.* (2017a) | Convective weather | Modified Dijkstra's algorithm | 2D discussed | Enroute/ within 20 minutes before departure | Partial: considers ATC |
| Sauer *et al.* (2017b) | Winter frontal system and air mass convection, including turbulence and icing | DIVMET geometric heuristic (Hauf *et al.* 2013) | 2D | Pre-departure and in-flight | Partial: considers ATC |

From Table 2.4, it could be seen that phenomenon investigated by most research was convective weather. With regards to dimension of avoidance, most of the papers supported avoidance manoeuvres in the horizontal plane (2D). The rerouting problem increases in complexity when more dimensions are considered. For the flight phase covered, many of the papers considered weather rerouting before take-off. In addition, most of the papers did not adequately integrated the avoidance of other aircraft into the rerouting process. Another important observation was that most existing research did not sufficiently consider the impact of the routing process on passengers.

## 2.9 Summary

This chapter has reviewed literature on flight rerouting during adverse weather. Various weather services, such as SIGMET and AIRMET, have been discussed. In addition, the serious effects of adverse weather was discussed. These include structural damage to aircraft, accidents and decreased manoeuvrability of aircraft, flight delays, increased pilot and ATC workload. From the literature reviewed, convective weather was a commonly investigated phenomenon. Most of the papers also supported avoidance manoeuvres only in the horizontal plane (2D). Weather avoidance was conducted mostly before take-off. This means such paper did not adequately consider weather avoidance under airborne conditions. In addition, most of the reviewed research did not sufficiently integrate the avoidance of other aircraft into the rerouting process. Another important observation was that most existing research did not sufficiently consider the impact of the routing process on passengers.

To ensure appropriate weather avoidance, different kinds of data are needed for the avoidance processes. These kinds of data are discussed in the next chapter, in addition to the distribution mechanisms required for distributing the data to relevant air transport stakeholders.

# Chapter 3 : AIR TRANSPORT DATA

## 3.1 Introduction

Airspace are complex entities with large number of interacting systems that need to communicate and share information with each other. A lot of data is being generated and exchanged in the air transport industry daily. These forms of data include data generated on-board the aircraft (on-board data), data exchanged between the aircraft and air traffic controllers (ATC data) and surveillance data (e.g. ADS-B). Other forms of exchanged data include aircraft operational communication (AOC) data for operational and maintenance purposes, aircraft passenger communications (APC), as well as environmental and weather data (like pressure and temperature). These types of data are often critical and real time, with stringent safety and regulatory requirements. However, the interacting systems use different communication and data technologies, and are distributed in space. It is also desirable that the modification of one system or domain should not adversely affect other systems. This implies that interoperability and loose coupling are highly desirable. One way of achieving these goals is to use middleware, which provides a common layer underneath applications, and enables the various platforms to interact seamlessly. A discussion on the various forms of data, middleware and associated communication mechanisms follows.

## 3.2 System Wide Information Management (SWIM)

Interacting parties in the air transport sector need to communicate with each other and share information. In the past, a party that needed some system functionality had to be directly connected to the providing bodies (Fig. 3.1). These interacting parties include air traffic control, airport operators, pilots and airline operating centres. This was not an efficient way of achieving connectivity. This led to the development of the System Wide Information Management (SWIM) framework. SWIM core components are loosely coupled, use service-oriented architecture (SOA) principles and open standards (SESAR JU 2016), see Fig 3.2. SESAR's SWIM-Supported by Innovative Technologies (SWIM-SUIT) and FAA's SWIM were some major projects set up in this area (Houdebert and Ayral 2010b; Houdebert and Ayral 2010a).

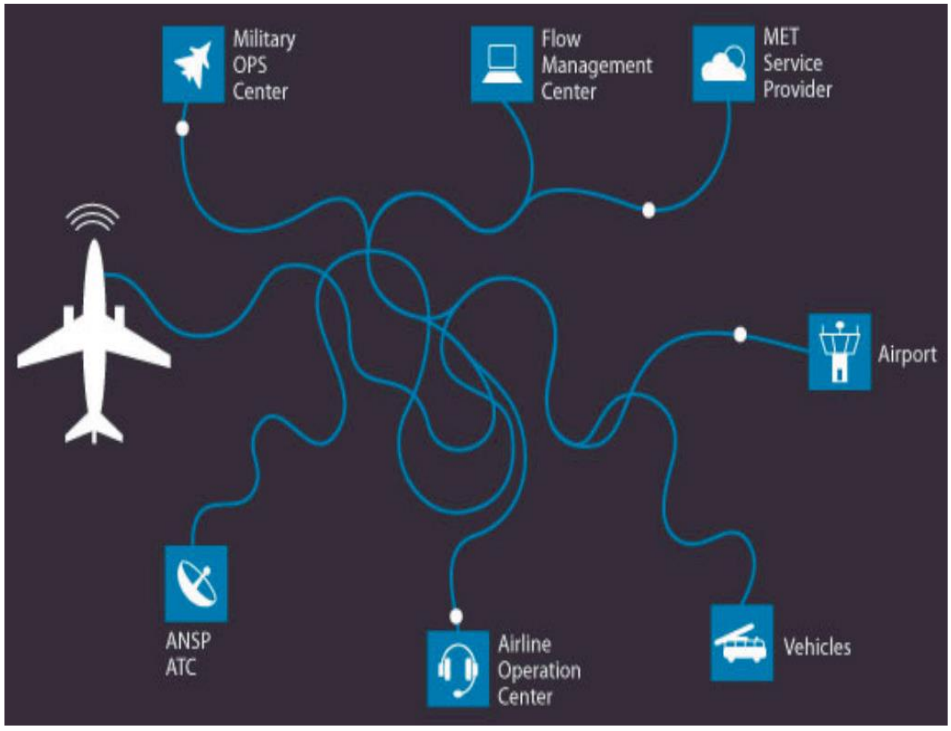


Fig. 3.1: Legacy approach to air transport information infrastructure (SESAR JU 2016)

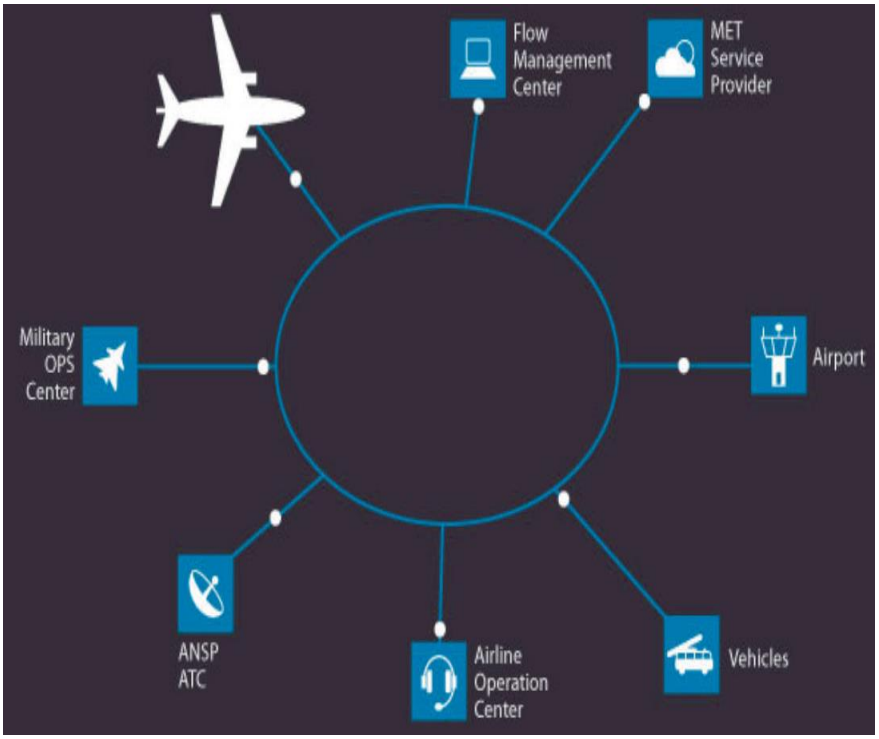


Fig. 3.2: System Wide Information Management (SWIM) approach (SESAR JU 2016)

A related issue was the development of unified data models, semantics and formats. In the past, for each type of air transport data, different formats were used by different bodies and countries. The recent trend, reflected in the SWIM, is to have a unified global data models. Examples of such models include AIXM, FIXM and IWXXM for aeronautical, flight and weather information respectively (Standley *et al.* 2012).

The SWIM Global Interoperability Framework can be divided into the following layers: Network connectivity, SWIM Infrastructure, Information Exchange, Information Exchange Services and SWIM-enabled Applications layers. The layers are shown in Fig. 3.3.

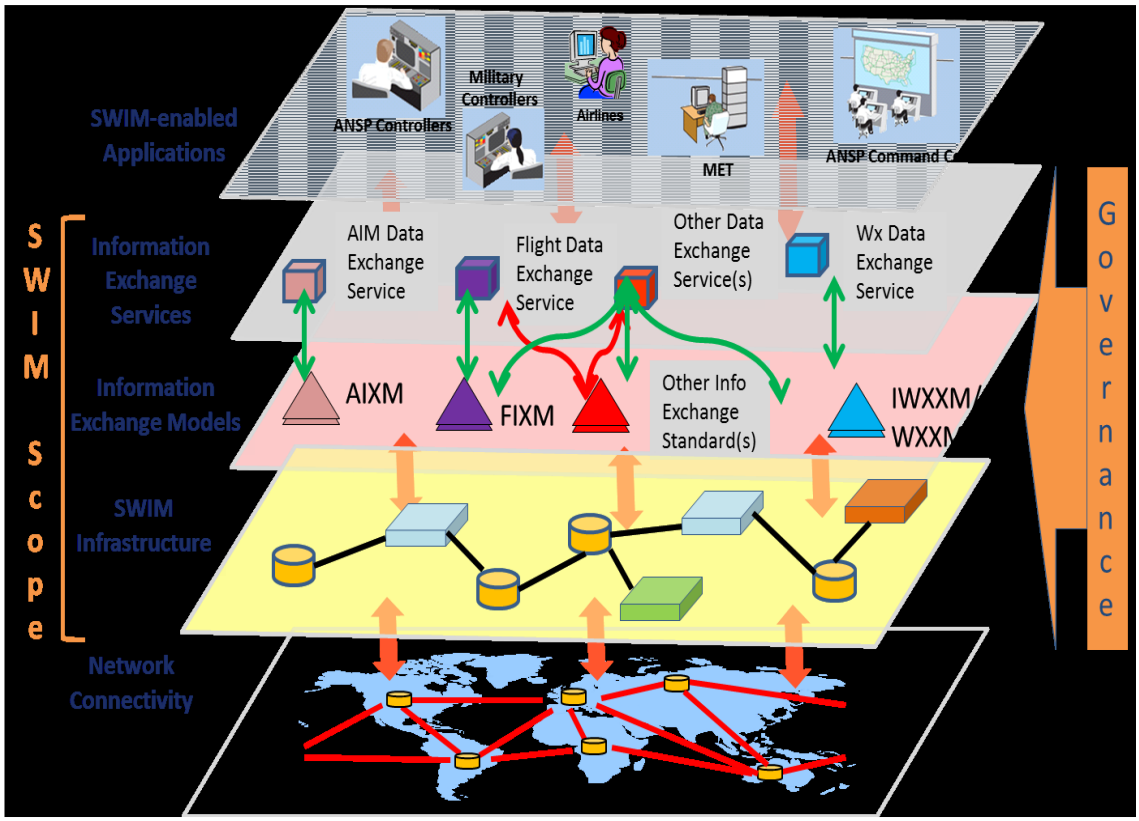


Fig. 3.3: Layers of the SWIM framework (ICAO 2017)

a. *Network Connectivity Layer* refers to the global IP-based telecommunication networks over which SWIM framework runs (ICAO 2017). The layer is responsible for delivering messages to their destinations. It also makes use of naming and addressing technologies like DNS.

b. The *SWIM Infrastructure* provides core services like messaging, enterprise service management, service-layer security, reliability, data representation and registry (using UDDI, for example) (ICAO 2017). The SWIM Infrastructure is made up of access points. A SWIM access point implements the core services and act as the point through which the consumer application connects to the SWIM network, see Fig. 3.4 for a functional diagram of a SWIM access point (ICAO 2017). Adapters are needed to connect legacy ATM systems to SWIM access points (Crescenzo *et al.* 2010).

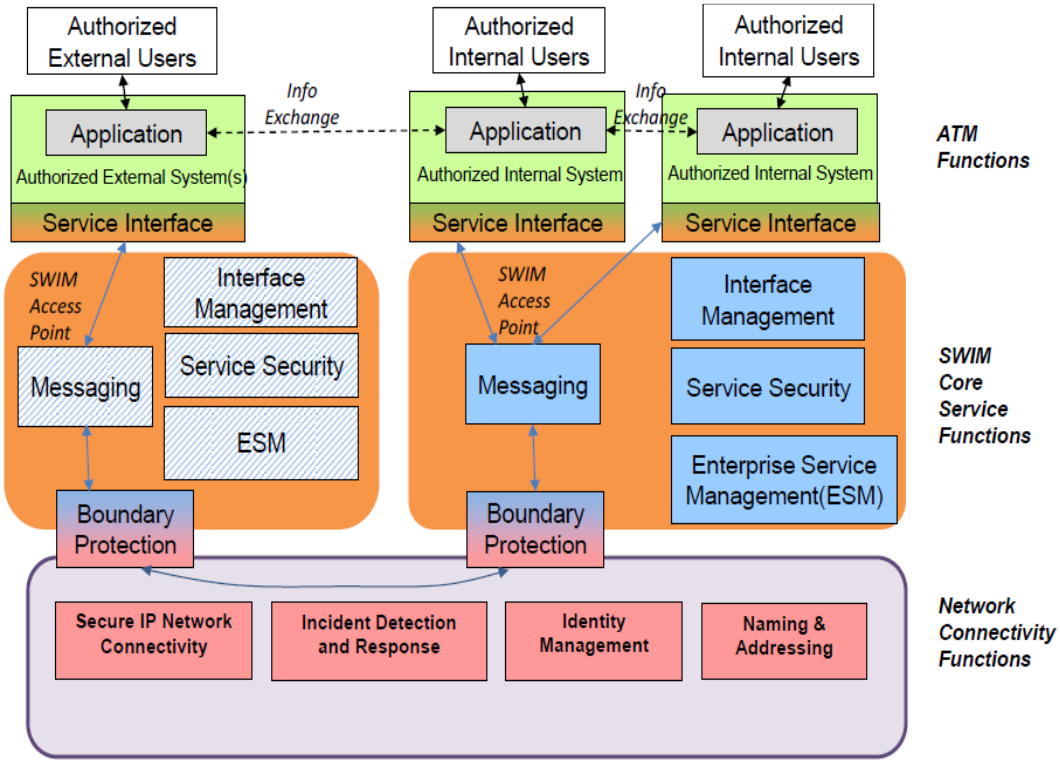


Fig. 3.4: Functional diagram of a SWIM access point (Crescenzo et al. 2010)

c. *Information Exchange Models layer*: This layer handles the modelling and description of data formats, structure and content (Crescenzo *et al.* 2010). For example, weather at a particular airport could be described using IWXXM. Other supported data models and schemas include AIXM, WXXM, FIXS and WXXS. The AIRM provides domain-specific semantic interoperability, which ensures an agreed set of meanings for key concepts.

d. *Information Exchange Services*: This layer provides technology-independent description of service characteristics (Crescenzo *et al.* 2010). Its role includes the definition of interfaces, using technologies such as WSDL, WADL, WFS and WMS.

e. *SWIM-enabled applications* are systems that are able to communicate with each other through the above-mentioned layers of the SWIM framework. These applications provide and consume ATM and other related information over the framework (Crescenzo *et al.* 2010). Such applications are used by airlines, meteorological centres, ANSPs and air traffic control.

## 3.3 Open Geospatial Consortium specifications

Some air transport data require geospatial services. Features of aeronautical facilities, such as airport location, can be described using geospatial models. The Open Geospatial Consortium (OGC) developed a set of web services-based standards for manipulating geospatial resources. The standards include Web Map Service (WMS), Web Feature Services (WFS) and Geography Markup Language (GML) specifications (Davis *et al.* 2009).

WMS supports the retrieval of server-rendered maps which are transferred to the client as images (e.g. PNG and JPEG) and text markup formats (e.g. XML) (OGC 2006; Davis *et al.* 2009). Such maps could be retrieved from geospatial databases (OGC 2006). In addition, WMS supports queries on map content. WFS provides a platform to perform operations and queries on features (Bermudez *et al.* 2009; OGC 2010). Features represent physical geographical entities, like rivers and roads. GML is used to encode responses, which are rendered on the client side(Davis *et al.* 2009).

One application of these services includes the GML-based AIXM which is a unified way of modelling aeronautical features such as aerodromes (Davis *et al.* 2009). In addition, AIXM-encoded features of aeronautical facilities could be retrieved using WFS (OGC 2012).

## 3.4 Types of air transport data

Important kinds of data are produced, used and exchanged in the air transport industry, including aeronautical data, weather data, surveillance data and flight

data. These data are important inputs to the weather avoidance process and are discussed as follows.

### 3.4.1 Aeronautical data

Aeronautical data refers to data about the actual navigation facilities and equipment in a given airspace, such as aerodrome locations and navigational aids (ILS, VOR etc). Most aviation authorities, like NATS publish aeronautical information manuals specifying this information (NATS 2017). Regular updates, deviations and warnings are also published in form of NOTAMs (notices to airmen).

Apart from the usual versions, electronic machine-usable formats of aeronautical data have become available. One initial problem was the plurality of formats available. Therefore, some major stakeholders, including EUROCONTROL, have defined a unified format: the Aeronautical Information Exchange Model (AIXM) format (Liu *et al.* 2011). AIXM was derived from the Geography Markup Language, an XML-based format defined by the Open Geospatial Consortium (OGC).

### 3.4.2 Weather/Meteorological data

Weather information is important for safe and efficient operation of air transport systems. For instance, adverse weather could make flights hazardous. Therefore, the timely distribution of weather information and forecasts is of much importance. Such data can be distributed using SIGMET (Significant Meteorological conditions), METAR or related reports (Daniels *et al.* 2012).

The defined format for METAR, SIGMET and similar products is traditional alphanumeric code (TAC). Whereas messages using these code are human readable, they are not as readily suitable for machine processing. To enhance the processing and exchange of weather data among diverse stakeholders, XML-based formats such as IWXXM have been defined (ICAO 2016a). Amendment 76 to Annex 3 of the ICAO regulations (the Convention on International Civil Aviation) encouraged the dissemination of weather data using models that are globally interoperable (WMO 2014). This involves the use of XML/GML-based formats and schema. Such XML-based data formats are machine-readable and

can be validated against specified schema. Fig 3.5 shows the components and dependencies of IWXXM (WMO 2016a).

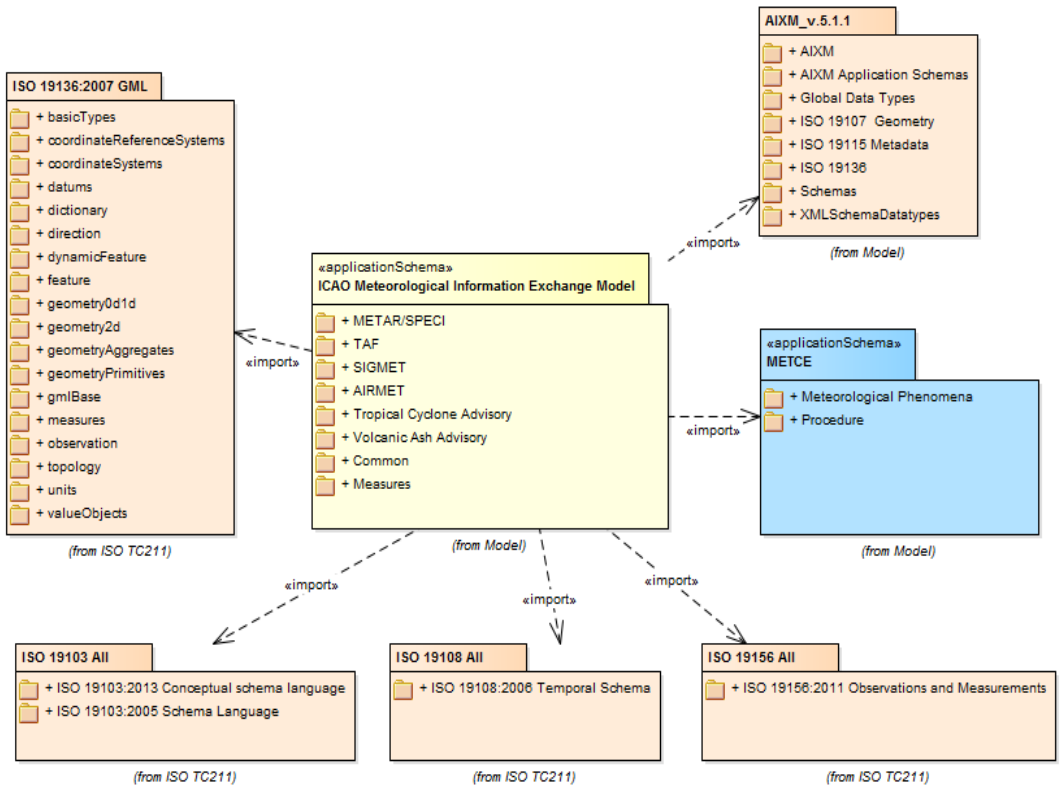


Fig. 3.5: IWXXM Package Diagram (WMO 2016a)

The IWXXM package provides models for the production and validation of meteorological reports such as METAR, SIGMET and AIRMET messages (WMO 2016a). IWXXM uses some elements of the AIXM data model. In addition, IWXXM uses standards produced by Technical Committee 211 (ISO TC211) of the International Organization for Standardization (ISO) (WMO 2014; WMO 2016a). These standards include ISO 19136:2007 or GML (BSI 2009), ISO 19103 and ISO 19108. The ISO 19103 standard defined guidelines for the unambiguous modelling of geographical information using a UML-based approach (BSI 2015). ISO 19108 considered temporal properties of geographical information, such as the temporal attributes, association and operations of features and metadata (BSI 2008).

The IWXXM package is also based on components such as the METCE (Modèle pour l'Échange des Informations sur le Temps, le Climate et l'Éau). A section of

METCE defines relevant forecast and observation types. Other parts of the standard present data models for metrological procedures and for phenomena such as tropical cyclone and volcanos (WMO 2014). METCE builds on the framework of ISO 19156:2011, an ISO standard that provides a schema for observations and associated sampling features (characteristics) (BSI 2013). Observations, as defined by the standard, assign numbers or symbols to phenomena using specified processes.

### 3.4.3 Surveillance data

To enhance efficient operation, air traffic management and collision avoidance, air spaces are monitored. Sources of surveillance data include ADS-B, primary radar, and secondary surveillance radar (SSR) (ICAO 2007b). Airborne SSR (e.g Mode S) transponders provide information, like aircraft identity and altitude, in response to interrogation by ground receivers. ADS-B is an important component of next generation air traffic management. ADS-B Out transponders broadcast messages indicating key information like aircraft location, velocity, and identity (Strohmeier *et al.* 2014). This information can be received by ADS-B In equipment on ground-based systems and other aircraft.

### 3.4.4 Flight data

The details of flights for an aircraft need to be communicated to many parties. For instance, the destination airport will need to know the estimated arrival of an aircraft for landing scheduling and other purposes. To facilitate the sharing of flight plans and trajectory, a number of standards have been developed, including the Flight Information Exchange Model (FIXM). The FIXM is XML-based and defines models for flight plan, emergency, aircraft, route and other relevant flight information (FIXM 2012; SESAR JU 2012) . Fig. 3.6 shows the packages of the FIXM logical model.

As seen in Fig. 3.6, the FIXM model has three main sub-divisions: messaging, base and flight (MIT 2017). The message package supports messaging requirements of flight and flow information (FIXM 2017a). It specifies message metadata and negotiation elements such as message type and unique message identifiers. The package also covers other supporting structural elements.

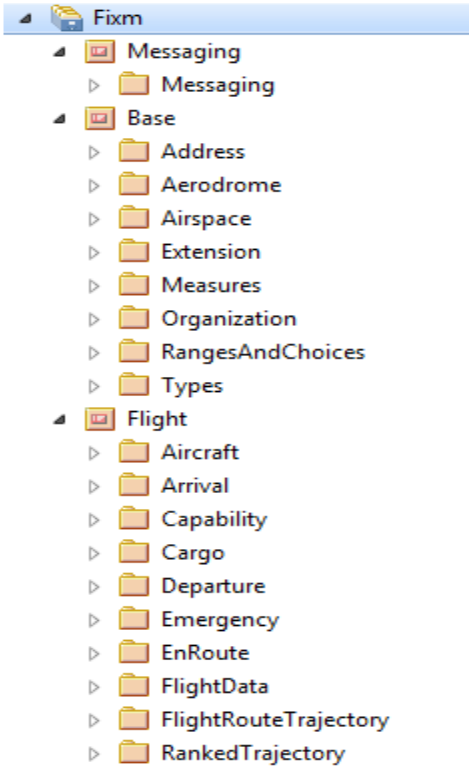


Fig. 3.6: Packages of FIXM logical model (MIT 2017)

The flight package covers flight and flow domain-oriented elements, as specified in the FIXM operational data description document (FIXM 2017b; MIT 2017). Elements that are part of the flight package include Aircraft, Arrival, Enroute, Departure, Emergency, Cargo and FlightData (FIXM 2017a). The characteristics of the aircraft used for a given flight can be modelled with the Aircraft sub-section. Details about the arrival and departure of a flight to and from an airport are contained in Arrival and Departure packages respectively. En-route information such as ATC coordination are described in the EnRoute package. Details of filed emergency during a flight are specified in the Emergency package. Details and constraints of a flight route are modelled in the FlightRoute package. FlightData package is central to the FIXM logical model and describes all the details of a flight (FIXM 2017a).

The base group of packages specifies low-level elements that are common to multiple FIXM packages (FIXM 2017a; MIT 2017). Its Address sub-section provides a model for contact information such as addresses. The Aerodrome package models aerodrome facilities like runways and aircraft stands. Airspace

package considers airspace characteristics such as significant points and ATS routes. Classes to be used for making extensions to FIXM are contained in the Extension package. Navaids package models information about navigational aids. The Measures sub-section considers the types and units of measurements relevant to flight data. The choices and combinations of these measurements are modelled in RangesAndChoices. Modelling of information about organisations is described in the Organization package. Models for basic numeric, temporal, geometry and textual types are considered in the Types package (FIXM 2017a).

## 3.5 Middleware for data exchange

An important enabler of air transport data exchange is middleware. Middleware lies between applications and the transport layer of the OSI protocol stack. It enables applications to access the networked resources transparently, almost as if they were local resources. The middleware layer handles many of the details of communications between nodes, so that the application developer does not need to focus on low-level details.

Middleware have various levels of abstraction, varying from low-level functions above the operating system (OS) to high-level application programming interfaces (APIs). Schmidt (2002) and Schmidt and Buschmann (2003) identified four middleware layers: host infrastructure middleware, distribution middleware, common middleware services and domain-specific services (Fig. 3.7).

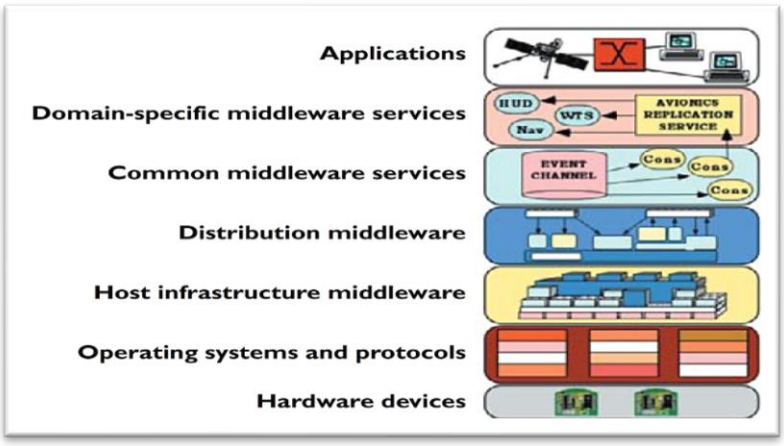


Fig. 3.7: Middleware layers (Schmidt 2002; Schmidt and Buschmann 2003)

*Host infrastructure middleware* abstracts low-level native OS programming mechanisms like POSIX pthreads to provide easy use of inter-process

communication, synchronisation and concurrency (Schmidt 2002). The layer improves portability of applications across operating systems. Examples of host infrastructure middleware include the Java Virtual Machine and Microsoft Common Language Runtime (CLR).

*Distribution middleware* run on top of host infrastructure middleware to provide higher-level abstraction and extended functionality (Schmidt 2002). The layer enables the programming of distributed applications without going into the details of the underlying hardware, operating system or communication network. By using this layer, for instance, interaction between components is done by calling functions/methods of the provided objects. Typical examples of such middleware include SOAP, Java Remote Method Invocation (RMI) and OMG Common Object Request Broker Architecture (CORBA) (Schmidt 2002).

*Common middleware services layer* provides groups of commonly used, domain-independent, high-level tasks in the form of services (Schmidt 2002). Such services include security, event notification, database connections, multimedia streaming and transactions. Therefore, programmers can focus on the core logic of their applications, instead of writing routine software functionality. The layer enables component re-use, as well as scheduling and coordination of resource use. Examples of middleware at this layer includes CORBA Common Object Services, Microsoft .NET and Sun Enterprise Java Beans (EJB).

*Domain specific middleware services layer* provides specialised functionality, targeting particular application domains, such as air transport, medical imaging, or telecommunications (Schmidt 2002). Since they address the needs of particular domains, they have the potential of speeding development of applications in those domains. However, they often have limited utility outside those domains and have the potential of duplicating functionalities across frameworks.

## 3.6 Messaging mechanisms and types

Communication and interaction between entities in a middleware-based system is carried out using a number of mechanisms and software patterns. These include request/reply, push/pull, and publish/subscribe, as discussed below.

### 3.6.1 Synchronous communication

In synchronous Request/Reply communication, a software thread sends a request and waits till it gets a reply (Fig. 3.8). The requesting thread is said to block while waiting for the reply. The disadvantage of this style is that a network disruption or long delay could cause the requesting application to crash and lose data (Roshen 2009). Both communicating parties have to be available at the same time. In addition, the requesting process is unable to do anything else while waiting for a reply. The main attraction of this method is its low processing overheads. Examples of middleware that support synchronous communication include Remote Protocol Call (RPC) and Remote Method Invocation (RMI).

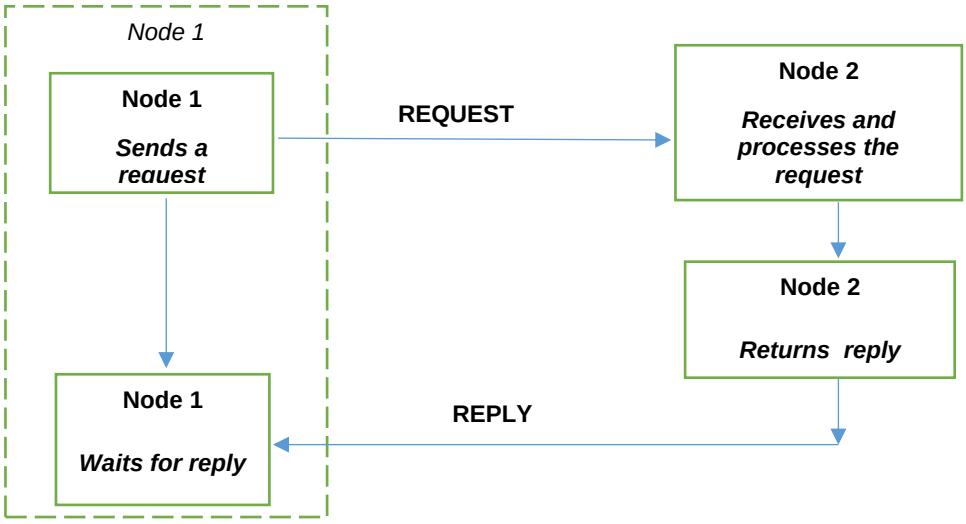


Fig. 3.8: Synchronous Request/Reply communication

### 3.6.2 Asynchronous communication

The aforementioned disadvantages of synchronous communication are addressed by using asynchronous messaging. Instead of directly communicating with receivers, a message producer sends messages to a queue that is accessible to the receivers. Receivers can then access the messages from the queue, possibly at a later time. Message senders and receivers are said to be loosely coupled. The sender thread can send the message to the queue and continue to other tasks. In addition, the receivers do not have to be available at the same time the sender sends the message. In some implementations, the senders do not need to know who their receivers are.

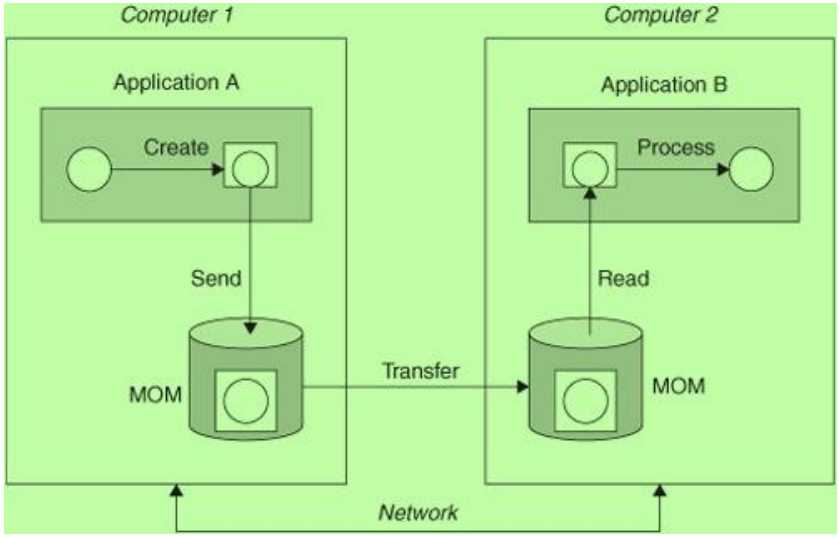


Fig. 3.9: Asynchronous messaging (Roshen 2009)

Asynchronous messaging could be *point-to-point* or *publish/subscribe* (Shahnaz *et al.* 2012). In point-to-point, communication is between one sender and one receiver at a time (Fig. 3.9). The sender sends messages to a queue and the application at the receiving end accesses ('pulls') it from the queue. Many Message Queue (MQ)-based middleware, such as AMQP and JMS, support both point-to-point and publish/subscribe messaging.

### 3.6.3 Publish/subscribe

In publish/subscribe messaging, key concepts include publishers, subscribers and topic. Information of a particular kind is represented using a data model or *topic*, identified by name. Parties interested in such data indicate their interest in the topic (i.e. subscribe to the topic). Whenever data is available, the data producer (*publisher*) sends data to the topic, and the middleware delivers the data to the interested entities (*subscribers*) (Eugster *et al.* 2003). Usually, the data is 'pushed' to interested parties (Roshen 2009). The publishers and subscribers do not need to know each other (space decoupling) nor be connected at the same time (time decoupling) (Eugster *et al.* 2003).

Some publish/subscribe implementations have intervening entities known as brokers, which provide directory services and act like message servers (iMatix 2012). Publishers send messages to the broker, while the broker is responsible

for forwarding the messages to subscribers. Brokers can hold published messages and make them available when subscribers come online. The disadvantage of centralised brokers is that they can become bottlenecks in the system. An alternative is to use peer-to-peer communication, where a publisher directly communicates with its subscribers (iMatix 2012). However, this may mean that messages stored by the publisher are lost if the publisher becomes unavailable (that is, no durability).

### 3.6.4 Push/pull

Messaging interaction could be done in either pull or push mode. In pull mode, the data user (or client) specifically requests for data from the data provider (or server) (Medjahed 2008; Li *et al.* 2013). When it receives the request, the data provider replies with the requested data. In one-to-many systems, the replies could be sent using unicast, broadcast or multicast mechanisms. While they are similar to synchronous messaging, some pull systems support intervening message queues, which allows some message durability and loose coupling between the communicating parties. An example of pull messaging is a web client accessing media from a HTTP server. The disadvantage of the pull technique is that it incurs more messaging and polling by the client, compared with push methods.

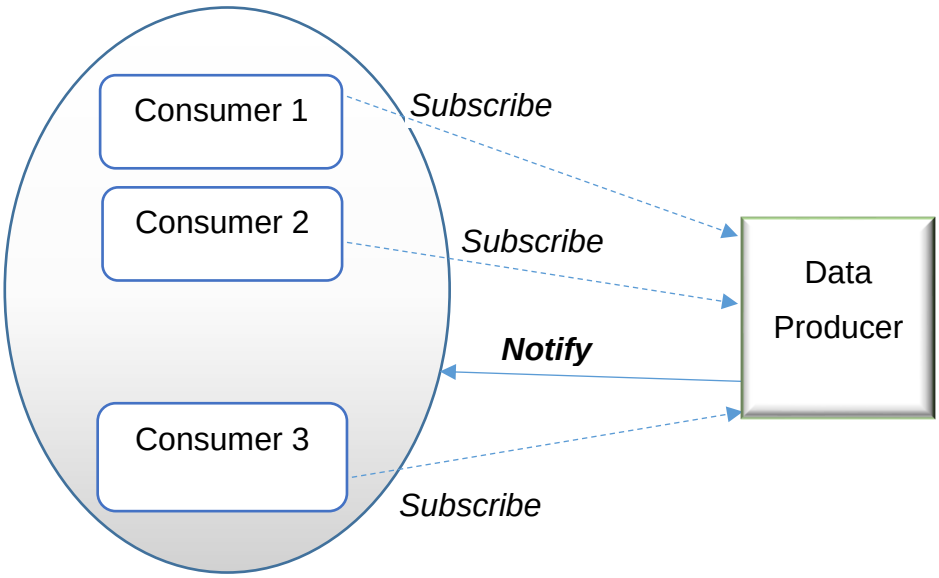


Fig. 3.10: Push messaging (Medjahed 2008)

In push systems, clients do not have to make explicit requests for data before being sent data by producers (or servers), see Fig 3.10. The data producer sends data (or event notifications) to consumers whenever an event happens. In some variants of push messaging, the producers send the message through brokers which deliver messages to consumers. In publish/subscribe forms of push messaging, consumers indicate their interest to receive data either directly to the publisher or through a broker (Warren *et al.* 2014). Push messaging could take place over unicast, multicast or broadcast. Examples of middleware that support push messaging include MQTT and JMS (Warren *et al.* 2014).

Some systems use both techniques. For example, the server could use push notifications to indicate availability of particular content. Interested clients then explicitly pull the actual content, e.g. video clips, from the server.

## 3.7 Middleware classes

Middleware can be grouped into different classes, based on the approach used to communicate between entities and kinds of operations supported. These classes are discussed in the following sections and include Remote Procedure Call (RPC), message-oriented middleware (MOM), service-oriented middleware (SOM), as well as object- and component-oriented middleware.

### 3.7.1 Remote Procedure Call (RPC)

RPC-based middleware enable a process to invoke remote program procedures (or functions) as if they were running on the same node as the process. Input parameters and return values are marshalled (packed or serialised) and sent over the connecting network to the remote node (Issarny *et al.* 2007). The remote node un-marshals (unpacks) it and executes the code. RPC relies on sockets, associated port numbers and IP addresses. In its original form, RPC uses synchronous communication, which leads to limited scalability and reliability (Roshen 2009). Applications built using RPC middleware are tightly coupled. However, RPC-based messaging is simpler and tends to have low processing overheads.

### 3.7.2 Object-oriented middleware

This class of middleware uses the object-oriented (OO) methodology and improves code re-use by exposing system functionality in form of objects (Issarny *et al.* 2007; Roshen 2009). Applications running on a node can invoke methods of remote objects running on another node. The OO approach enables application developers to make use of OO tools in developing middleware-based applications. This could make it easier and faster to develop and debug applications. One disadvantage of this approach is the increase in incurred overheads. Examples of such middleware include Microsoft DCOM, Java Remote Method Invocation (RMI) and Common Object Request Broker Architecture (CORBA) (Eugster *et al.* 2003).

### 3.7.3 Service-oriented Architecture (SOA) and Web Services (WS)

In SOA, the functionality of an application is decomposed into *services*. Services are self-contained, described in a standardised language and have well-defined interfaces. In addition, the implementation of a service is separated from its interface definition (Papazoglou and van den Heuvel 2007). Service-oriented architecture ensures the loosely coupling of distributed systems, as opposed to the tight coupling of the object-oriented paradigm (Schneider 2010).

Web services are well-known implementations of the SOA concept. The layers of an application utilising web services, based on that defined by W3C, are shown in Fig. 3.11 (W3C 2004). At each of this layer, a number of solutions and standards have been proposed. A discussion of these solutions follows.

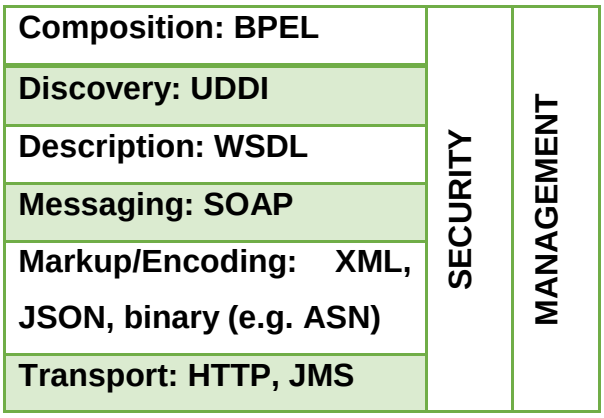


Fig. 3.11: A WSDL/SOAP-based web service stack (W3C 2004)

Web services, especially as defined by W3C, were predominantly Extensible Markup Language (XML)-based. XML defines resources in terms of (nested) tags and parameters. These components could be defined in a template file known as an XML schema. XML is a markup language that is human-readable and can also be processed by computers. Another advantage is that it is well-established and supported by many programming languages. However, its major disadvantage is that it is verbose, which contributes to large files size, complexity, excessive transmission overheads and bandwidth consumption. This disadvantage has adversely affected legacy web services protocols which are mostly XML-based, making adoption slow and difficult. XML has inspired many other markup languages designed for specific applications, such as the Geography Markup Language (GML) for geospatial data.

JSON (originally, JavaScript Object Notation) is a compact format that encodes data objects in name:value pairs (ECMA 2013). An array is represented by an ordered list between square brackets. Although it is language independent, the notation is similar to object literals in Javascript. JSON has many advantages over other data formats: it is lightweight, human-readable and easy to parse, especially when using JavaScript (ECMA 2013; Niu *et al.* 2014).

Binary formats encode data in machine readable form. For example, the Java programming language has methods for serialising/de-serialising objects in binary formats (writeObject() and readObject() methods respectively) (Maeda 2012). Other binary encoding formats include ASN, Apache Avro and Google Protocol Buffers. ASN.1 is an ITU-T standard which can be used to define the abstract syntax (or types) of data elements (Mundy and Chadwick 2004; ITU-T 2008). For example, FlightTime::=TIME means type FlightTime is assigned an element of the type TIME at runtime. The actual encoding is based on related standards such as Basic Encoding Rules (BER), XML Encoding Rules (XER) and Packed Encoding Rules (PER). For a data element, BER encoding produces a byte stream featuring its Type, Length and Value (TLV). ASN.1 has found application in telecommunications systems. Since data is converted to binary form before transmission, data processing is faster for ASN.1-based systems, compared with using XML (Mundy and Chadwick 2004). Its low bandwidth usage

makes it attractive when data resources are at a premium. Its major disadvantage is its low readability by humans.

Table 3.1 compares the various types of data encoding formats with respect to their unique characteristics.

Table 3.1: Comparison of data encoding formats

| **Encoding/ Feature** | **Verbosity/ Data size** | **Adoption** | **Human readable** | **Comments** |
|---|---|---|---|---|
| XML | High | Well established | Yes | Eases interoperability, debugging, maintenance |
| JSON | Low | Good | Yes | - |
| Binary, e.g ASN.1/BER | Very low | Established | No | Difficult for interoperability and troubleshooting |

Communication (messaging) between web services can be done using defined protocols. A well-established messaging protocol is SOAP (used to mean Simple Object Access Protocol). It is XML-based and can run on top of transport mechanisms like SMTP, HTTP and message queue (MQ) protocols (Curbera *et al.* 2002). The structure of a basic SOAP message consists of an XML element (envelope) with header and body child elements (Curbera *et al.* 2002).

To make use of a service, there needs to be a way to describe the functionalities the service can provide. One standard that addresses this issue is Web Service Description Language (WSDL), a W3C standard. WSDL (Christensen *et al.* 2001) defines services in terms of abstract endpoints (ports) and bindings to implementation in an actual protocol/message format. Such concrete formats include MIME, SOAP 1.1 or HTTP GET/POST. One disadvantage of WSDL is that it is XMl-based, with the usual data size and processing overheads.

### 3.7.4 Representational state transfer (REST)-ful web services

The original (SOAP/WSDL-based) web services implementation is complex, bulky and has a lot of overhead. Therefore, it is slow and consumes a lot of bandwidth. Developing applications using the stack is also difficult. The RESTful architecture (Fielding and Taylor 2002) has, therefore, been adopted in recent

times. RESTful services commonly access uniform addressable resources using HTTP operations. Messaging between RESTful elements is based on ‘Request/Respond’: resources are explicitly requested for and then provided by the party that receives the request. RESTful applications are stateless, which means that the server does not keep states of the client. The result of a given call should be independent of previous calls. One major advantage of REST/HTTP is its relatively low resource utilisation. In addition, the HTTP protocol is supported by most computing platforms.

### 3.7.5 Message-oriented middleware

Message-oriented middleware (MOM) is one way of ensuring loose coupling between systems. MOM entities communicate using messages, which are asynchronous (i.e. blocking is not necessary to receive a reply to a request) (Menasce 2005). Two mechanisms associated with MOM are message queues and publish/subscribe (Jia *et al.* 2014). By using message queues, a sender can send messages to one receiving party (Menasce 2005). An example of a message-queue protocol is AMQP. However, in publish/subscribe systems, a node (the publishers) sends data to a number of interested parties subscribing to a topic. Standards supporting the publish/subscribe paradigm include DDS and Java Message Service (JMS).

#### *3.10.5.1 Java Message Service* (*JMS*)

JMS defines a standardised abstract API for communication between MOM-based systems. JMS, now in its second version, supports both point to point (using message queues) and publish/subscribe messaging (Hapner *et al.* 2013). The major entities in a JMS application include providers, clients, destinations and ConnectionFactory, JMSContext, JMSProducer and JMSConsumer. Clients receive and send messages. Alongside the usual messaging capabilities, a provider is responsible for administration in a JMS application. ConnectionFactory is an interface for creating a connection (JMSContext). The JMSProducer and JMSConsumer objects are used for sending and receiving messages respectively. JMS has some support for quality of service (like persistent/non-persistent, durable/non-durable connections) (Chen and Greenfield 2004; Hapner *et al.* 2013). JMS is closely associated with the Java

programming language, and can be done in Java SE or Java EE (containers). However, many implementations add support for other languages.

#### *3.10.5.2 Advanced Message Queue Protocol (AMQP)*

AMQP is an OASIS middleware standard originally designed for business applications. Unlike JMS, it defines a wire protocol which helps interoperability. In addition, data types and message format are extensively defined by the standard. However, it does not define an actual API. It supports publish/subscribe and message queue messaging (O'Hara 2007). AMQP uses a peer-to-peer binary transport protocol, which was chosen because it targets high performance applications (O'Hara 2007; OASIS 2012). The protocol is, however, specified in XML to ease code generation (O'Hara 2007). For the underlying transport layer, protocols supported include TCP/IP, SCTP and UDP (O'Hara 2007). The standard provides support for the Transport Layer Security (TLS) Protocol and the Simple Authentication and Security Layer (SASL). Types of AMQP nodes include producers, consumers and queues which generate, receive and store/forward messages respectively. Containers (brokers and clients applications) can contain a number of these nodes. The standard also supports transaction messaging, as well as message priority, filtering and durability. Examples of actual implementation of the standard include OpenAMQ, RabbitMQ and Apache Qpid (Subramoni *et al.* 2008; Fernandes *et al.* 2013).

#### *3.10.5.3 Data Distribution Service for Real time Systems (DDS)*

DDS is a high-performance publish/subscribe middleware standard released by OMG. The standard (OMG 2007) consists of two layers: a mandatory Data-Centric Publish-Subscribe (DCPS) layer and an optional upper Data Local Reconstruction Layer (DLRL). The platform independent model (PIM) of DDS DCPS defines the concept of a distributed global data space into which objects called publishers send data. Entities that have indicated interest in the data (subscribers) are able to receive it. The entities interact using data models known as topics. Each topic has a name, quality of service (QoS) parameters and a data type (OMG 2007). Subscribers receive data, which they pass on to interested applications. The DLRL layer provides direct access to DCPS features through the use of objects that are mapped to DCPS entities (OMG 2007).

A Real time Publish Subscribe protocol (RTPS)-based wire protocol was introduced to ensure interoperability between different DDS vendors (OMG 2007; OMG 2010). It uses the OMG Common Data Representation (CDR) format to convert Interface Definition Language (IDL) data types to actual octet streams transmitted on the wire, guaranteeing interoperability (OMG 2011). The PIM of RTPS defines four modules (Structure, Discovery, Messages and Behavior). RTPS communication entities, such as Writer and Reader endpoints, are defined by the Structure module. The Discovery module obtains information about remote entities through the use of predefined (built-in) topics. The Simple Participant Discovery Protocol (SPDP) and Simple Endpoint Discovery Protocol (SEDP) are used for discovering participants and endpoints respectively. The supported message exchanges between endpoints and the associated timing requirements are defined by the Behaviour module. The Messages module gives the actual format and possible contents of the messages. RTPS implements reliability on top of the transport layer. In addition to the PIM, it defines a PSM (platform specific model) for UDP/IP. Its low latency overheads and extensive support for QoS make DDS stand out. It has applications in aerospace and industrial automation.

Table 3.2 shows a comparison of the middleware technologies discussed above, in terms of key properties such as messaging style, performance and QoS support.

Table 3.2: Comparison of some middleware technologies

| ***Property/ Technology*** | **SOAP/WSDL-based web services** | **RESTful web services** | **DDS** | **JMS** | **AMQP** |
|---|---|---|---|---|---|
| **Architectural elements** | Client/Server | Client/ Server | Publishers, Subscriber, Topics | Providers, Clients, Producer, Consumer | Brokers, Queues, Producers, Consumers |
| **Messaging paradigms** | Request/ Respond | Request-respond, stateless | Publish/ Subscribe | Publish/ Subscribe, Message queues | Message queues, Transactional |
| **Resource use** | High | Lightweight | Low | Medium | Low |

Table 3.2: Comparison of some middleware technologies (cont'd)

| ***Property/ Technology*** | **SOAP/WSDL-based web services** | **RESTful web services** | **DDS** | **JMS** | **AMQP** |
|---|---|---|---|---|---|
| **Vendor interoperab-ility** | Yes, it uses popular XML messages | Yes, it often uses popular HTTP messages | Yes (using RTPS wire protocol) | No, it's an API | Yes, defines wire protocol |
| **Language support** | Mostly XML | Language independent | Multiple, e.g. C/C++, Java | Java-centric | Java (JMS syntax), C++, Python APIs etc (Vinoski 2006) |
| **Adoption** | Medium | High (Meng et al. 2009) | Medium | High | Medium |
| **Latency** | High, because of verbose messaging | Less than in WSDL-based WS (Upadhyaya et al. 2011) | Very low: binary protocol | Low | Low |
| **Quality of Service Support** | Not part of core specifications: WS-Reliable-Messaging | No, needs to be added | Very detailed | Partial | Limited |
| **Security** | WS-Security (Serme et al. 2012) | Not defined, implementation-specific/rely on security on other layers (Serme et al. 2012) | DDS-Security (beta standard) (OMG 2014) | Not defined | Part of main specification (TLS, SASL) |
| **Examples of Implementa-tion** | JAX-WS | JAX-RS | OpenDDS, OpenSplice DDS, RTI Connext | IBM WebSphere | OpenAMQ, RabbitMQ, Qpid (Subramoni et al. 2008; Fernandes et al. 2013) |
| **Typical applications** | General web applications | General web applications, Internet of things | Critical applications in real time systems, aerospace. | Enterprise applications | Real time financial transactions |

## 3.8 Summary

This chapter has discussed air transport data, paying particular attention to the sources and kinds of data applicable in this field. These kinds of data include aeronautical data, meteorological/weather data, surveillance and flight data. Some research projects in this area has also been reviewed. These include

System Wide Information Management-Supported by Innovative Technologies (SWIM-SUIT), which attempted to improve data exchange among stakeholders by providing an interoperable SOA-based communication framework. SWIM-SUIT also supported unified data models and formats. The Open Geospatial Consortium (OGC) has produced web services-based standards for working with geospatial resources, such as the Web Map Service (WMS) specifications. WMS is one of the server-side standards for deploying aerodrome map services. Middleware classes and message patterns have also been discussed.

# Chapter 4 : MODELLING OF ADVERSE WEATHER AVOIDANCE

## 4.1 Introduction

This chapter presents a model for the avoidance of adverse weather. During adverse avoidance, several competing factors need to be considered and the appropriate decisions will be made. However, the complexity and number of parameters inherent in the process make the problem of weather avoidance very challenging. Existing models have often adopted an isolated approach that address each of the mitigating factors individually. These factors include flight delay, operational costs, and passenger convenience. In addition, the impact of modified short-term routes on passengers has not been adequately addressed in literature. An integrated model to address these challenges is proposed in this chapter.

## 4.2 Problem definition

Given the graph structure of the airspace under consideration, an optimal flight path is required from a start point to a destination airport. Fig. 4.1 shows a network graph of an airspace. A set *V* represents the waypoints and airports (nodes) in the airspace. A flight link $l_{ab}$ (in set *E* of links) exists between two waypoints *a* and *b* if there exist at least one route that passes through the two waypoints. There are costs $J_1 \ldots J_n$ associated with the link. Candidate paths are created as sequences of waypoints through which the aircraft passes.

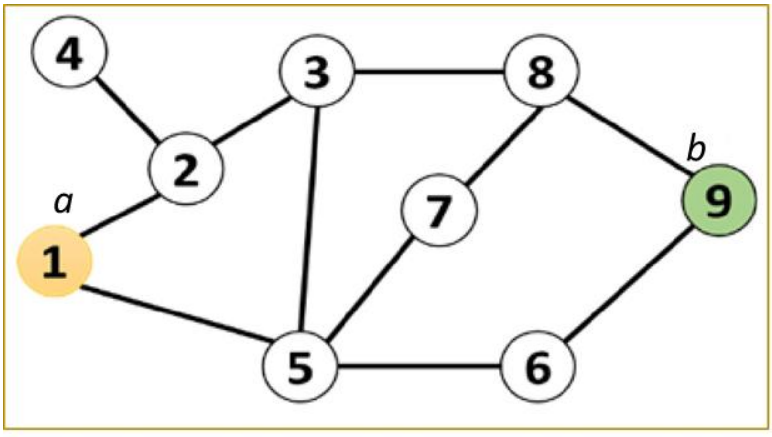


Fig. 4.1: Network graph of waypoints

While the developed model can be applied to most flight phases, the short term en-route scenario is considered. In the case considered, the waypoints do not have to be on the same plane. Each trajectory waypoint is fully described by a set of latitude, longitude, altitude and time values. The time value represents the estimated or actual time the aircraft passes through the trajectory location defined by the latitude, longitude and altitude values.

The scenario considered the existence of one or more regions of adverse weather in the area considered. For example, Fig. 4.2 shows adverse weather regions around waypoints 3 and 7. The goal is to find an optimal path from a start waypoint *s* to destination *d* that avoids the regions of adverse weather, while minimising costs $J_1$, $J_2$, …, $J_j$. These costs include delay costs, weather impact costs, fuel costs and cost of missed connections. $J_j$ is the $j^{th}$ cost considered. The constraints to the problem are given by $\{C_1, C_2, \ldots, C_c\}$. In this case, the minimisation problem can be written as:

$$\text{Minimise } J_t = \{ J_1, J_2, \ldots, J_j \}$$

$$\text{subject to } \{C_1, C_2, \ldots, C_c\} \qquad (4.1)$$

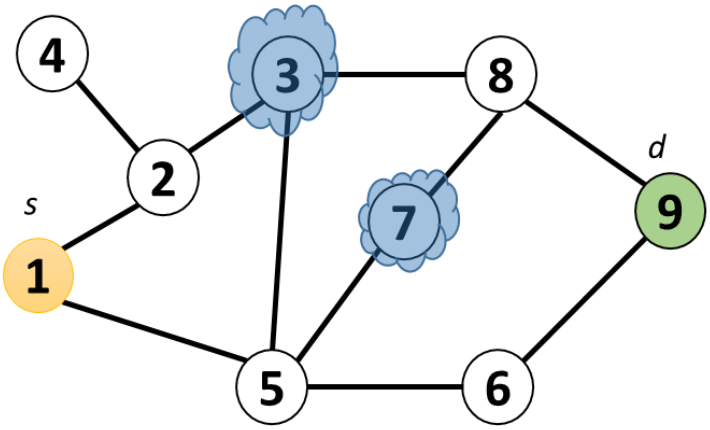


Fig. 4.2: Network graph showing regions of adverse weather

The constraints $\{C_1, C_2, \ldots, C_c\}$ determine if a proposed solution is feasible. One constraint is that the aircraft needs to maintain a safe distance from the weather region. Another constraint is that the aircraft needs a minimum separation distance from other aircraft. Other constraints are discussed in Section 4.6.

A key objective of airlines is high customer satisfaction. Adverse weather can seriously affect this objective, by triggering flight delays, missed connections and other inconveniences. Most existing work on rerouting flight paths do not adequately consider the direct impact of weather on passengers. Therefore, a new objective known as the passenger inconvenience factor (PIF) has been defined to address this. The goal of the designed algorithm is to minimise this objective. In addition, we consider operational costs such as fuel cost. Other operational costs such as crew costs are assumed to be negligible in the short-term case considered. If the number of aircraft considered is given by $N_i$, $J_1 = J_{AOC}$ and $J_2 = J_{PIF,}$ equation (4.1) becomes,

$$\text{Minimise } J_t = \{J_{AOC}, J_{PIF}\}$$

$$\text{subject to } C_1, C_2, \ldots, C_c \tag{4.2}$$

where $J_{AOC}$ is the aircraft additional operating costs and $J_{PIF}$ is the impact of the flight path reroute on passengers.

## 4.3 Assumptions

For the model, it is assumed that the speed on each flight link is constant. In addition, the average flight time of each flight link is known. The flight time values can be derived from historical data, for example. The flight stage focussed on by the model is the en-route phase. Landing and take-off times considered to be constant. Initial schedules of flights need to be specified. Also, it is assumed that air traffic sectors and their capacities are known.

## 4.4 Derivation of the passenger inconvenience factor

The inconvenience experienced by passengers is caused by a number of factors such as number of connections, flight delay, intensity of weather phenomenon and the type of adverse weather and changes in flight level (Ayo 2017; Ayo *et al*. 2017). In order to derive the total PIF, the weighted sum of the component costs was obtained. The advantage of using weights is that the relative importance of the considered costs can be dynamically set by airlines to fit their priorities.

Hence, the passenger inconvenience factor ($J_{PIF}$) is given by

$$J_{PIF} = \sum_{p=1}^{Np} (w_{Nc}J_{Nc} + w_m J_m + w_\delta J_\delta + w_{FL}J_{FL}) \tag{4.3}$$

where $J_{Nc}$ = cost of number of connections, $w_{Nc}$ = weight assigned to connection cost, $w_m$ = weight of the weather impact cost, $w_\delta$ = weight associated with the delay cost, $w_{FL}$ = weight associated with the flight level change cost, $J_m$ = the weather impact cost, $J_\delta$ = flight delay costs, $J_{FL}$ = flight level change cost, and $N_p$ = the total number of passengers. The costs are discussed and derived below.

### 4.4.1 Flight delay costs

Adverse weather or its avoidance can lead to delays or late arrival of aircraft. This often leads to serious inconvenience to passengers, due to missed appointments etc. The flight delay cost is calculated as:

$$J_\delta = \delta . C_\delta \tag{4.4}$$

where delay $\delta$ is defined by the difference (in hours) between actual arrival time and expected time of arrival, and $C_\delta$ = unit cost of delay. In this work, $C_\delta$ was given a value of $2 (Arıkan *et al.* 2016) and can be further adjusted by airlines. The actual arrival time is the sum of the actual departure time and the actual flight time. To obtain the total flight time, the duration of each flight leg is summed up, as shown in the following equation (Ahn and Ramakrishna 2002; Wu *et al*. 2007; Zhang *et al.* 2014).

$$F_{Ti} = \sum_{a=s}^{d} \sum_{\substack{b=s \\ b \neq a}}^{d} (c_{ab}^{T} . T_l(a,b)) \tag{4.5}$$

$c_{ab}^{T}$= is a decision variables for flight duration, that is equal to 1 if the route of the aircraft passes through link (*a, b*) and 0 otherwise (Zhang *et al.* 2014). $F_{Ti}$ is the total flight time.

$F_{Ti}$ is approximated by dividing the distance $D_l(a,b)$ of each flight leg by the average velocity $v_l(a,b)$ of the aircraft during that leg (Wu *et al.* 2007). The resultant values are summed to find the total flight time. For each flight leg, the time taken by the aircraft is:

$$T_l(a,b) = \frac{D_l(a,b)}{v_l(a,b)} \tag{4.6}$$

The leg distance $D_l(a,b)$ between any two points *a* and *b* on a route can be calculated using great circle distance. The longitudinal and latitudinal coordinates of waypoints were used. Using Haversine formula (Beauducel 2012; Veness

2017), where $\varphi_a$ and $\varphi_b$ are the latitudes of points *a* and *b* respectively, $\lambda_a$ and $\lambda_b$ are the longitudes of points *a* and *b* respectively and $E_R$ is the earth's radius, let

$$\alpha = sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_a \,.\cos\varphi_b \;\; sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{4.7}$$

$$c = 2.atan2\left(\frac{\sqrt{\alpha}}{\sqrt{1-\alpha}}\right) \tag{4.8}$$

then,

$$D_l(a,b) = E_R.c \tag{4.9}$$

### 4.4.2 Cost of missed connections

For flights consisting of multiple flight legs, it is desirable that each of the component legs arrive on time. One of the consequences of delayed flights is the inconvenience that comes when passengers miss their connecting flights. Missed connections happen when the sum of the actual arrival time of the preceding flight leg and the minimum connection time is later than the departure time of the connecting flight. Actual arrival time is obtained by adding flight time to the actual departure time of the flight. The inconvenience to passengers in this scenario is measured by missed connection costs, defined by:

$$J_{Mc} = \; N_{Mc}.C_{Mc} \tag{4.10}$$

where $J_{Mc}$ = total connection cost, $N_{Mc}$ = number of missed connections and $C_{Mc}$ = cost per missed connection. $C_{Mc}$ was given a value of $50, similar to Arikan *et al.* (2016).

### 4.4.3 Weather impact cost

The weather impact cost $J_m$ calculates the direct impact of the weather on passengers. It is dependent on the severity of the weather and the cost associated with the weather type, as shown in the equation below.

$$J_m = \; \sum_{m=1}^{Nm}(I_m \,.C_m) \tag{4.11}$$

where $N_m$ = total number of weather cells, $I_m$ = intensity of the $m^{th}$ weather cell and $C_m$ = the cost associated with the type of weather cell. This cost is determined by how seriously the weather could affect an aircraft and its passengers. For example, severe turbulence has high impact on an aircraft and its passengers (Sermi *et al.* 2015).

To calculate the value of $C_m$, the scenario is considered in which the weather is so bad that a passenger is delayed and had a missed connection. In that case, the missed connection cost of \$50 is incurred. That is, $I_m.C_m$ = \$50 and $C_m$ = \$50/$I_m$. If such severe weather has the highest intensity value of 100%, this means that $I_m$ = 100% = 1. Hence, $C_m$ = \$50/1 = \$50.

### 4.4.4 Cost of flight level changes

During adverse weather avoidance, it may be necessary to change flight level in order to avoid regions of adverse weather. However, this may impact passengers' flight experience. The level of inconvenience due to flight level changes is modelled by cost,

$$J_{FL} = N_{FL}.C_{FL} \tag{4.12}$$

where $J_{FL}$ = total flight level change cost, $N_{FL}$= number of flight level changes and $C_{FL}$ = cost per flight level change. $C_{FL}$ was given a value of \$50 by considering a situation where the avoidance of a flight level change leads to a missed connection and associated costs.

Substituting equations 4.4, 4.10, 4.11 and 4.12 in 4.3 gives the derived PIF as:

$$J_{PIF} = \sum_{p=1}^{Np} [w_{Nc}.N_c.C_{Nc} + w_m.\sum_{m=1}^{Nm}(I_m.C_m) + w_\delta.\delta.C_\delta + w_{FL}.N_{FL}.C_{FL}] \tag{4.13}$$

## 4.5 Aircraft operational costs

An important set of costs incurred during adverse weather avoidance include operational costs. Such operational costs are crew costs, maintenance costs and fuel costs. In this work, it is assumed that in the short-term scenario no additional crew costs was incurred, apart from those already planned in the original flight. In addition, the aircraft maintenance cycle was not impacted and hence no additional costs was incurred. Fuel costs as a result of the weather avoidance process can be determined from aircraft performance databases. Fuel cost as a function of distance can be estimated using linear regression on the performance data (Park and O'Kelly 2014). Fuel costs per passenger $F_{Bm}$ *for* a distance $D_m$ is given by:

$$F_{Bm} = \frac{a+b.D_m}{Npi} \quad (4.14)$$

where *a* and *b* are constants that are characteristic of the regression line, and $N_{pi}$ is the total number of passengers on an aircraft *i*. Additional fuel costs for movement to distance $D_n$ is given by:

$$J_B = F_{Bn} - F_{Bm} = \frac{a+b.D_n}{Npi} - \frac{a+b.D_m}{Npi} = \frac{b.(D_n - D_m)}{Npi} \quad (4.15)$$

Hence,

$$J_B = \frac{b.\Delta D}{Npi} \quad (4.16)$$

To obtain the value of $b/N_{pi}$, the fuel consumption data of an aircraft was considered. Fig. 4.3 shows the variation of fuel consumption with distance for an A320 aircraft (EEA 2016).

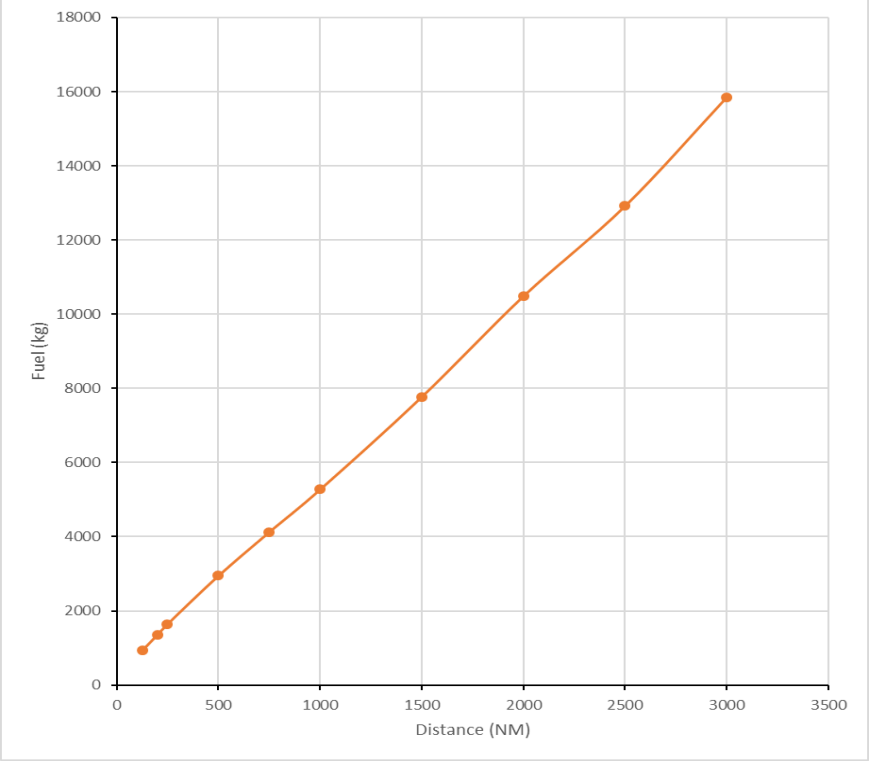


Fig. 4.3: Fuel consumption for aircraft. Data source: EEA (2016).

Applying an approximation using linear regression to the fuel data, we get total fuel *y* used in kilogrammes for a flight as:

$$y = 291.13 + 5.1063\ D_m \quad (4.17)$$

The coefficient of determination $R^2$ is a measure of how well the line of best fit fits the data. It varies from 0 (no fit) to 1 (perfect fit). The $R^2$ value for the fuel data in Fig. 4.3 was 0.9994, indicating a good line fitting.

To get the cost of fuel per passenger, consider equation 4.17. The equation gives the mass $y$ of fuel used (in kg). To get the monetary cost of this mass of fuel, multiply it by the unit fuel cost $k_f$ ($/kg). In addition, to get the cost per passenger, further divide by number of passengers $N_{pi}$. That is, both sides of equation 4.17 are multiplied by $k_f$ * /$N_{pi}$, giving:

$$\frac{k_f}{N_{pi}} y = \frac{k_f}{N_{pi}} \left( 291.13 + 5.1063\, D_m \right) \tag{4.18}$$

Substituting a unit fuel cost of $k_f$ = $0.6736/kg (IATA 2018), and assuming the number of passengers $N_{pi}$ to be 180, equation 4.18 becomes:

$$\frac{k_f}{N_{pi}} y = 1.0895 + 0.0191\, D_m \tag{4.19}$$

It can be observed that the left hand side of equation 4.19 is the fuel cost per passenger, which is $F_{Bm}$. Therefore, comparing right hand sides of equations 4.14 and 4.18, $b$ is obtained as 0.0191/passenger-NM. Since 1 NM = 1.8520 km, $b$ is equivalent to $0.0103/passenger-km.

Substituting expressions (4.13) and (4.16) into (4.2) and using a weighted sum approach, the overall objective is derived as,

$$Minimise\, J_t = \sum_{p=1}^{Np} \left[ w_{Nc}.N_c.C_{Nc} + w_m.\sum_{m=1}^{Nm}(I_m\,.\,C_m) + w_\delta.\delta.C_\delta + w_{FL}.N_{FL}.C_{FL} + w_B \frac{b.\Delta D}{Npi} \right] \tag{4.20}$$

To obtain the overall cost $J_{to}$ for a set of multiple aircraft, the cost incurred by each aircraft for a given set of alternative routes are summed up. The minimisation problem in this case is:

$$Minimise\, J_{to} = \sum_{i=1}^{Ni} J_t \tag{4.21}$$

By substituting expression (4.20) in (4.21), the expression for the overall cost is obtained as:

$$Minimise\ J_{to} = \sum_{i=1}^{Ni} \sum_{p=1}^{Np} [w_{Nc}.N_c.C_{Nc} + w_m.\sum_{m=1}^{Nm}(I_m.C_m) + w_\delta.\delta.C_\delta + w_{FL}.N_{FL}.C_{FL} + w_B\frac{b.\Delta D}{Ni}\ ] \quad (4.22)$$

## 4.6 Constraints

To ensure acceptability by pilots and air traffic control, solutions developed by the model must meet certain conditions. One of such constraints is to ensure minimum separation from other aircraft. In addition, the aircraft should maintain a safe distance from the weather phenomena. The speed of the aircraft should be within the range specified by its manufacturer. The constraints for the above model are given below, and determine what reroute solutions are feasible.

1. $D_p(i,j) \geq D_{sep}$ (4.23)

   $D_p(i,j)$ is the distance between aircraft *i* and *j* located at ($x_i$, $y_i$, $z_i$) and ($x_j$, $y_j$, $z_j$) respectively at time *t*, and is estimated as:

   $$D_p(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \quad (4.24)$$

   $D_{sep}$ is the minimum separation distance.

   This constraint implies that the distance among any two aircraft must be more than the minimum separation distance to keep aircraft safety.

2. $D_w(i,w) \geq D_{w(sep)i} + R_w$ (4.25)

   where $D_w(i,w)$ is the distance between aircraft *i* at point ($x_i$, $y_i$, $z_i$) and the weather phenomenon with centre at ($x_w$, $y_w$, $z_w$) and radius $R_w$. $D_{w(sep)i}$ is the recommended minimum separation between the aircraft and the adverse weather, for example 10 nautical miles (NM) in case of thunderstorms (NATS 2010). $D_w(i,w)$ is given by:

   $$D_w(i,w) = \sqrt{(x_i - x_w)^2 + (y_i - y_w)^2 + (z_i - z_w)^2} \quad (4.26)$$

   This requirement ensures that each aircraft maintains a safe distance from the weather phenomena (Wu *et al.* 2007).

3. $v_{wl}(t) \leq v_{wmax}(t)$ (4.27)

The speed of the aircraft, that is $v_{wl}(t)$, should be less than or equal to the maximum possible for the weather phenomenon. This covers situations, such as reduced visibility, in which the aircraft reduces its speed because of the weather.

4. $v_{min}(t) \leq v_l(t) \leq v_{max}(t)$. (4.28)

The speed of an aircraft should be within the range specified by its manufacturer, that is, its speed $v_l(t)$ at any time during a flight should be between the minimum operating speed $v_{min}(t)$ and maximum speed $v_{max}(t)$.

5. $n_k(t) \leq n_{max}(t)$. (4.29)

The number of aircraft $n_k(t)$ in a sector $k$ should be less than the maximum $n_{max}(t)$ allowed at that time. This ensures that ATC workload is appropriate.

6. $c_{ab}^T = \{0, 1\}$. (4.30)

$c_{ab}^T$ is the decision variable for flight duration. The value of each variable is 1 if the aircraft passes through link ($a$, $b$) and 0 otherwise.

7. $\sum_{\substack{b=s \\ b\neq a}}^{d} c_{ab}^T - \sum_{\substack{b=s \\ b\neq a}}^{d} c_{ba}^T = \begin{cases} 1 & if\ a = s \\ -1 & if\ a = d \\ 0 & otherwise \end{cases}$ (4.31)

This constraint specifies link (flight leg) characteristics for each way point (Ahn and Ramakrishna 2002), with respect to a given route of an aircraft. The first condition ($a$=$s$) indicates that only one link should leave the start node. Similarly, only one link should end at the destination node ($a$=$d$). For intermediate nodes, the last condition holds: the difference between the number of links entering a waypoint and the number of links leaving it should be zero.

8. $\sum_{\substack{b=s \\ b\neq a}}^{d} c_{ab}^T = \begin{cases} \leq 1 & if\ a \neq d \\ = 0 & if\ a = d \end{cases}$ (4.32)

This constraint (Ahn and Ramakrishna 2002) indicate that the sum of links leaving the destination waypoint should be zero for a given route of an aircraft. For other nodes, the sum of links leaving the node should be less than or equal to one.

## 4.7 Summary

This chapter presented an improved model for the weather avoidance process for aircraft. The model is passenger-centric and adequately considers the impact of the avoidance process on passengers. The model simultaneously considered the impact of path reroutes on aircraft operational costs. The factors covered by the model included flight delay, weather impact, missed connections, fuel costs and flight level changes.

# Chapter 5 : GENETIC AND FIREFLY ALGORITHMS FOR ADVERSE WEATHER AVOIDANCE

## 5.1 Introduction

The avoidance of adverse weather often involves the rerouting of flight paths. To produce such alternative paths, rerouting algorithms such as genetic algorithms have been proposed (Alam *et al.* 2006; Kai-quan *et al.* 2015). However, the avoidance process needs to be done efficiently and consider multiple factors, such as passenger inconvenience. This is important to minimise costs and impact on passengers. This chapter presents improved algorithms for the generation of alternative flight routes. The algorithms also support the enhanced rerouting model developed in this work.

## 5.2 Genetic algorithm for alternative route generation

One way of generating alternative flight paths during adverse weather avoidance is by using techniques such as genetic algorithms. Genetic algorithms have improved ability for global search of the solution space. However, the existing genetic algorithms for optimal path generation can get trapped in some local optimals. To solve this problem, a modified algorithm was developed using a composite mutation strategy. One feature of the developed framework is that it can receive data from global aviation data platforms such as SWIM. This is done using middleware such as RESTful systems. The functional diagram of the algorithm, including its inputs, is shown in Fig. 5.1 and discussed below.

### 5.2.1 Flight rerouting layer

The flight rerouting layer is the higher layer end-user application. It is made up of the rerouting algorithm, configuration parameters and visualisation modules. The rerouting module is the main algorithm that detects adverse weather regions, generates the flight path reroutes, based on the data inputs and configuration parameters. The configuration parameters module stores the default and user-defined preferences for the platform. These preferences include the values for parameters of the rerouting algorithms, such as the unit delay and missed connection costs. The visualisation module presents graphical display of the

results from the rerouting module. The visualisation module receives reroute data from the main simulation module and airspace data from the end user adapters.

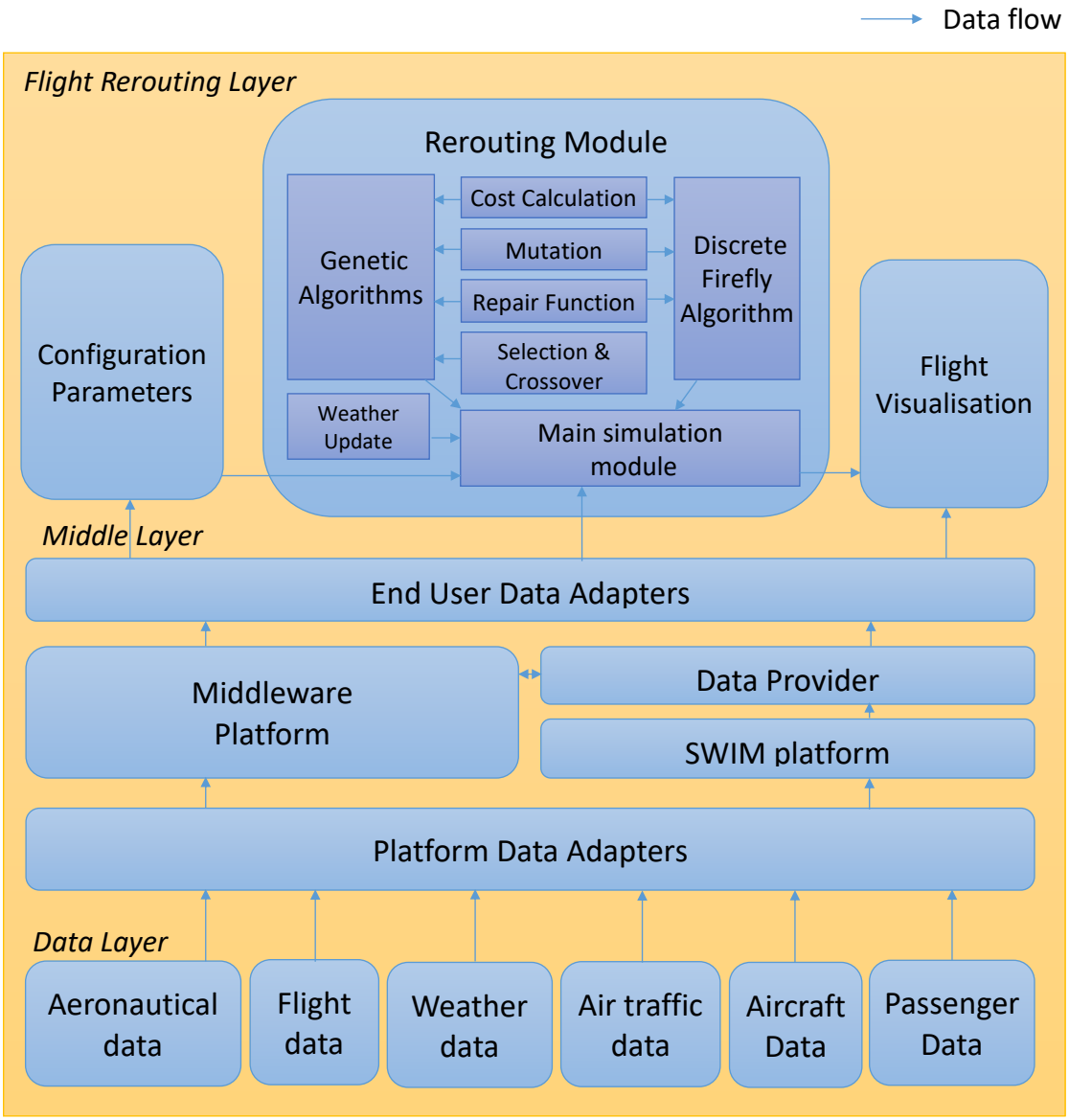


Fig. 5.1: Architecture of flight path reroute framework

The components of the rerouting layer include the main simulation algorithm which executes the rest of the flight rerouting layer. The main simulation algorithm acquires the required aeronautical and flight data using the end user data adapters. Other inputs to the main simulation module include the configuration parameters, data from the weather update module, and results from the genetic and discrete firefly rerouting algorithms. The outputs of the main simulation

module are sent to the flight visualisation module. The weather update module is responsible for processing current weather information. The processing involves determining which waypoints are affected by the weather. The data from the weather update module is sent to the main simulation module.

The genetic algorithms (GA) rerouting module uses GA-based techniques to generate alternative flight paths. The genetic algorithm makes use of results of the cost calculation, mutation, repair function, selection and crossover sub-functions. The sub-functions are discussed in the next section of this chapter. Because of this modular architecture, each of these sub-functions can be modified separately. Similarly, the discrete firefly rerouting (DFA) algorithm module uses a DFA-based method to produce flight path reroutes for adverse weather avoidance. The module obtains data from the cost calculation, repair and mutation modules.

### 5.2.2 Middle and data layers

The middle layer of the reroute framework is responsible for the acquisition and distribution of data. The end user data adapters convert data to formats that are compatible with the components of the flight rerouting layer. For example, if the application in the flight rerouting layer can only work with a binary data format, such as arrays, the end user data adapter converts the data to this format. The end user data adapters obtain data directly from the middleware data distribution platform. In addition, the adapters can obtain data indirectly through third-party data providers.

The middleware platform module supports the acquisition and proper distribution of aeronautical, flight and weather data using middleware such as REST-based frameworks. The data provider is a third-party entity that is responsible for the acquiring the relevant data from their sources and distributing them to application owners, such as airlines. As shown in Fig. 5.1, the data providers acquire the data over generic middleware platforms or using standardised SWIM platform interfaces. Both the SWIM platform and middleware platform obtain the relevant kinds of data through platform data adapters.

The platform adapters convert air transport data to formats suitable for distribution by the SWIM and generic middleware platforms. The capabilities of the adapters

include the conversion of data from TAC formats to XML-based formats, such as IWXXM. The inputs to the platform adapters come from the data layer modules. The data layer contain the data sources for aeronautical, flight, weather, air traffic, aircraft and passenger data. These data sources include airports, air traffic controllers and air navigation service provider.

## 5.3 Pseudocode of the rerouting algorithm (GA-based)

The rerouting module uses a genetic algorithm-based approach to derive a feasible reroute with the least cost. The reroute is given by a shortest path that minimises the total costs. The pseudocode of the core algorithm (Fig. 5.2) is given below, from the point of view of an aircraft. Fig. 5.3 shows the flowchart of the algorithm. The major steps of the algorithm are also explained below.

```
Start
        Identify start and end waypoints
        Read updated weather data
        For each weather cell
                Update airspace network
        End For
        Generate initial population
        While (termination criteria not met):
                Compute fitness of route population
                Select individuals for next generation
                Cross over routes
                Mutate routes
                Repair routes
                Replace parents with children routes
        End While
        Visualise flight path reroute and costs
End
```

Fig. 5.2: Flight path rerouting algorithm (GA-based)

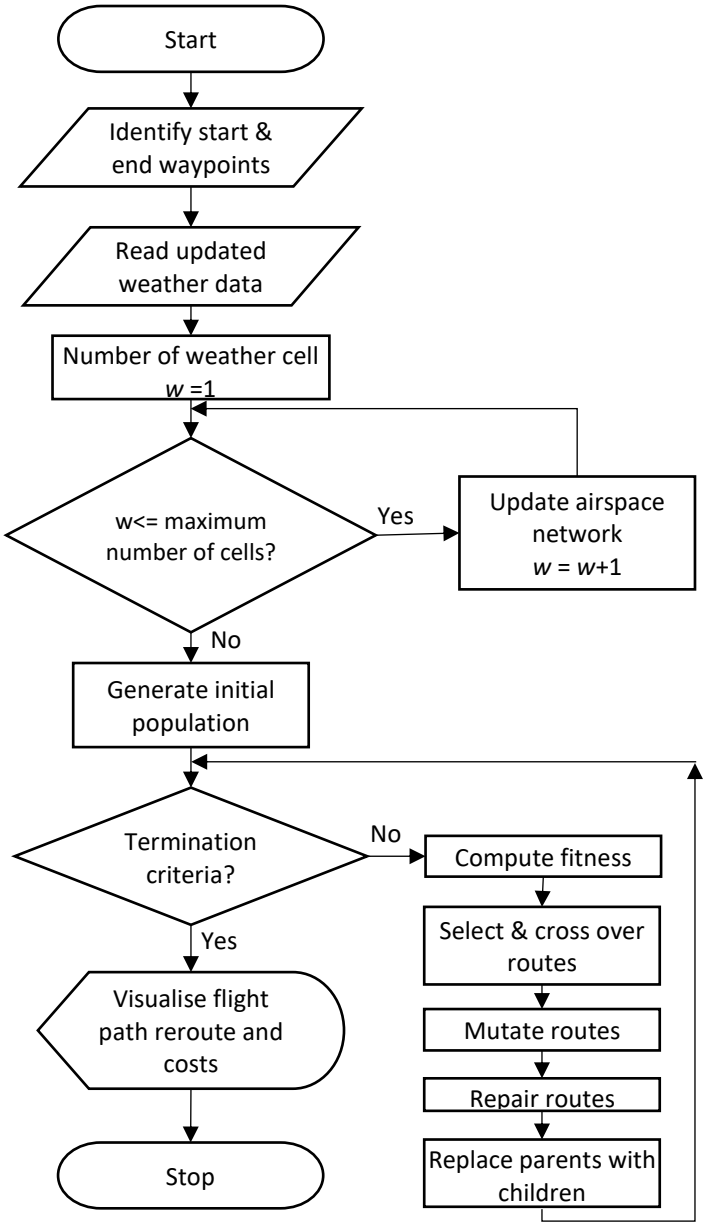


Fig. 5.3: Flowchart for flight path rerouting algorithm (GA-based)

### 5.3.1 Chromosome structure

Each candidate solution is represented by a chromosome. A chromosome is encoded as a series of waypoints that make up the considered routes. The chromosomes can have variable lengths. Fig. 5.4 illustrates an example of a chromosome, where *a, b, p* and *q* represent waypoints. In this work, waypoints are represented by integer values (their identification numbers). The waypoints are listed in order from the start waypoint to the destination.

| *a* | *b* | *p* | *q* |
|---|---|---|---|

Fig. 5.4: An example of a chromosome structure

Waypoints include locations of navigational aids and fixes derived using multiple navigational aids. In addition, for a given flight path, each waypoint is associated with a time value that indicates the time the aircraft would arrive at the waypoint.

### 5.3.2 Weather data update

This part of the algorithm acquires current weather data and checks to see if there are any adverse weather in the airspace considered. If there are adverse weather regions, then the airspace network structure is updated as required. Weather impact costs are added to the impacted flight legs. Affected waypoints are located at distances less than the minimum weather separation distance from weather regions. The weather data sources include SIGMET, TAF and METAR reports.

### 5.3.3 Generation of initial population

This function is used to obtain the initial set of possible routes (candidates). The start waypoint, destination and adjacency matrix are given as inputs. The adjacency matrix indicates if a path exists between any two waypoints. Initially, the start waypoint is taken as the selected waypoint. From the adjacency list of the selected waypoint, another waypoint is randomly chosen. The newly identified waypoint becomes the selected waypoint. This process is repeated until the destination node is reached. The list of selected waypoints form a route from the start waypoint to the destination waypoint.

### 5.3.4 Computing of fitness function

The fitness value of a route is a measure of its desirability. The goal of the algorithm is to find the path with the least costs. That is, paths with lower costs have greater fitness. Hence, the fitness value is taken as the inverse of the total costs incurred by the route. The total costs are calculated using the developed model (expression 4.20). The fitness function is given as:

$$Fitness\ F_t = \frac{1}{J_t} \tag{5.1}$$

### 5.3.5 Selection

During this stage, candidate routes (parents) are selected based on their fitness. In tournament selection, the population is randomly divided into sets of *n* individuals. From each set, the individual with the best fitness is chosen. For tournament selection without replacement, the sets of individuals are non-overlapping (Ahn and Ramakrishna 2002). In binary tournament selection, *n* equals two.

Another approach to selection is roulette wheel selection (Lee *et al.* 2007). In this method, the probability that a candidate route is selected is proportional to its fitness. Individuals are sorted according to their fitness values. A random number is generated and the first route with a cumulative fitness sum that is greater than the number is chosen. The process is repeated until the required number of parents are selected. Roulette wheel selection was adopted for the proposed algorithm. The use of roulette wheel selection ensures that diversity is retained across generations.

### 5.3.6 Crossover

A custom one-point crossover method was used, described in Algorithm 4.2. Crossover is done with a probability equal to a defined crossover rate. For two selected parents, their chromosomes are searched for waypoints that are common to both parents (Ahn and Ramakrishna 2002). One of the common waypoints is randomly chosen. The waypoints after this point for each parent is swapped with the corresponding set of waypoints in the second parent. This process is shown in the Fig. 5.5 and Fig. 5.6 below. The flowchart of the process is given in Fig. 5.7.

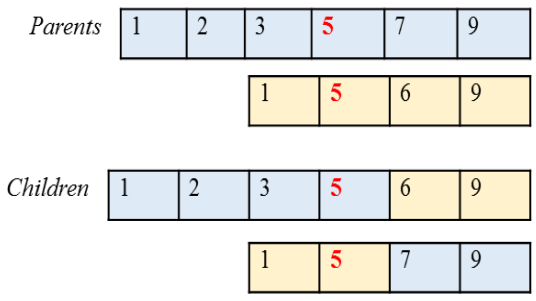


Fig. 5.5: An example of crossover process

Get both parents (*parent1* and *parent2*)

Get $C_w$ list of waypoints common to both parents

Randomly select a waypoint from $C_w$

(with loci *crs_pt1* and *crs_pt2* in *parent1* and *parent2* respectively)

Swap corresponding sections after the selected waypoint, i.e.

*child1 = parent1(1:crs_pt1)* joined to *parent2(crs_pt2+1:length(parent2))*

*child2 = parent2*(1:*crs_pt2*) joined to *parent1(crs_pt1+1:length(parent1))*

Fig. 5.6: Route crossover algorithm

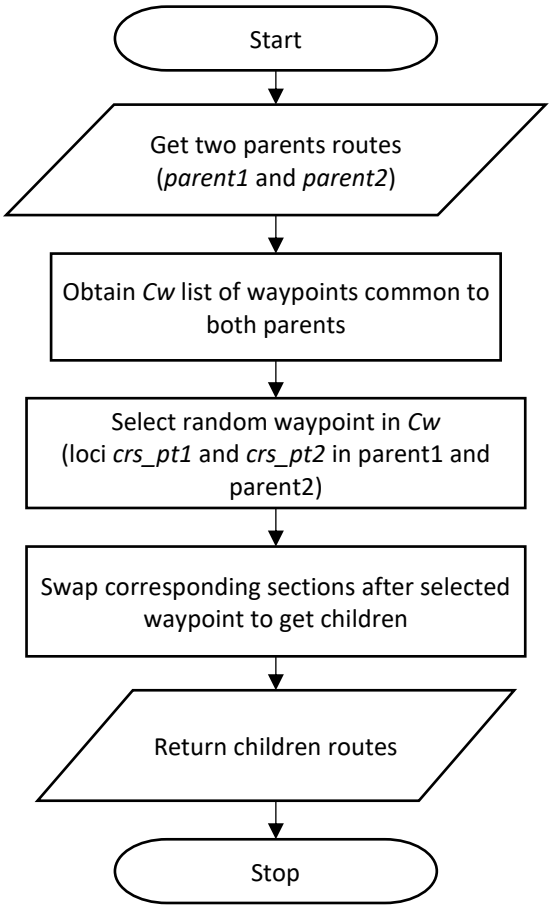


Fig. 5.7: Flowchart of crossover process

In Fig. 5.6, two selected parent routes are considered: *parent1* and *parent2*. The list $C_w$ is the list of waypoints that exist in both parents. A waypoint in this list is randomly chosen. The positions (loci) of the selected waypoint in *parent1* and *parent2* are *crs_pt1* and *crs_pt2* respectively. Waypoints from the first position (locus) to position *crs_pt1* on *parent1* are obtained. These waypoints are joined to waypoints from *crs_pt2* to the last position on *parent2* in order to produce one product *child1* of the crossover process. Similarly, waypoints from the first locus to locus *crs_pt2* on *parent1* are joined to waypoints from the first locus to locus *crs_pt1+1* on *parent2*.

### 5.3.7 Mutation process

During mutation, the candidate routes are modified in some way. This is done to increase exploration of the search space, while ensuring population diversity (Köksoy and Yalcinoz 2008). The mutation process is done with a probability equal to the mutation rate. In the mutation technique identified by (Ahn and Ramakrishna 2002), a node on a given path is randomly selected. Thereafter, nodes on the path that are before the selected node are removed from the topological database. A new initial path is then built from the selected point to the destination. The approach is given the identifier GAAR02. This method sometimes produced sub-optimal solutions, because the resulting algorithm gets trapped in local optima.

Another approach to mutation is to randomly select a locus in the considered chromosome. The gene at this locus is replaced by another waypoint that is randomly selected from the list of considered waypoints. This approach is known in this work as Mutation/Replace (GAARM). The repair function is run afterwards to remove loops and disconnections. The repair function is discussed in the next section.

To further improve the search ability of the rerouting algorithm, a modified mutation process was defined. This involves an insertion process, followed by a replacement procedure (Mutation/Replace). The genetic algorithm using this composite approach is known as GAIM16. The steps in the algorithm are shown in Figs. 5.8 and 5.9.

```
For each route in population
    If (random number ≤ mutation probability)
        Get random locus
        Select a random waypoint from considered area
        Insert selected waypoint after chosen locus
    End If
End For
/*Comment: Mutation/Replace follows*/
For each route in population
    If (random number ≤ mutation probability)
        Get random locus
        Randomly select a waypoint from considered area
        Replace gene at chosen locus with selected waypoint
    End If
End For
```

Fig. 5.8: Proposed route mutation algorithm

| | | | | | |
|---|---|---|---|---|---|
| **Original Chromosome** | 1 | 5 | 6 | 9 | |
| **Step 1: Random Insert** | 1 | 5 | 6 | **8** | 9 |
| **Chromosome from Step 1** | 1 | 5 | **6** | 8 | 9 |
| **Step 2: Replace** | 1 | 5 | **3** | 8 | 9 |

Fig. 5.9: Proposed mutation process

During the insertion procedure, a waypoint is randomly selected from the list of waypoints in the considered airspace. Next, a locus is randomly selected in the considered chromosome. The chosen waypoint is then inserted after the identified locus. Thereafter the replacement process is carried out. The repair function is run afterwards to remove loops and ensure there are no disconnections on the route. The flowchart for the algorithm is shown in Fig. 5.10.

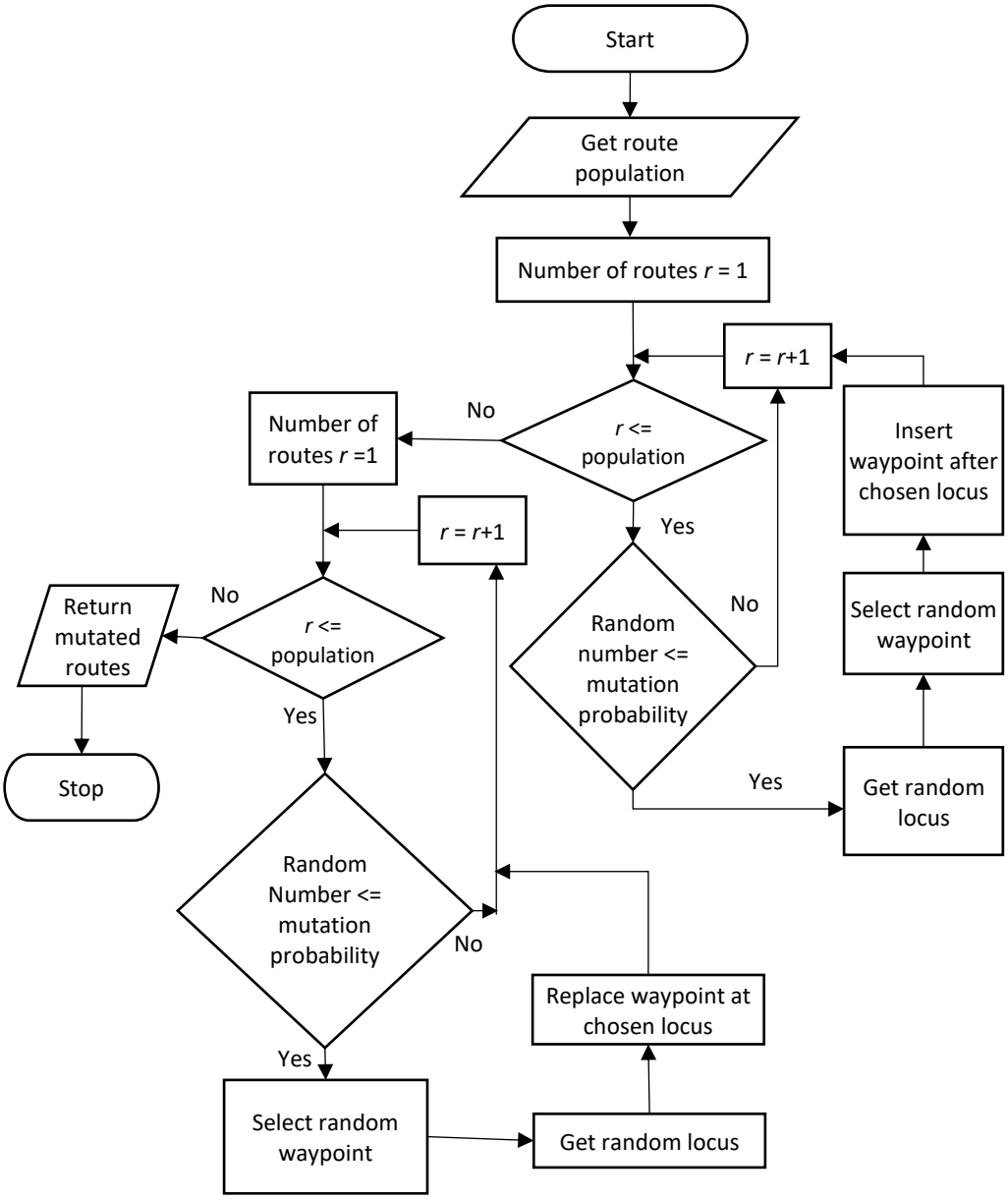


Fig. 5.10: Flowchart of proposed mutation technique

### 5.3.8 Repair function

During the application of crossover and mutation procedures, some infeasible routes may be generated. For example, the routes may contain loops. In addition, some of the routes may be disjoint. In addition, the repair function detects possible conflicts with other aircraft and attempts to resolve such situations. The repair function is shown in Fig. 5.11 and Fig. 5.12.

```
For each route in population
    While (route is disconnected) and attempts2 ≤ max_attempt2
        attempts2= attempts2 +1;
        Detect and repair aircraft conflicts
        Reconnect disjoint waypoints
    End While
    While (loop exists) and attempts < max_attempt1
        Remove loop
        attempts = attempts+1;
    End While
    If (route is empty or is still disconnected or loop exist)
    Get new path
    End If
End For
```

Fig. 5.11: Algorithm for repair function

As shown in Fig. 5.8, each route in the population is checked to determine if it is disconnected. In addition, waypoints that lead to conflict with the routes of other aircraft are detected and removed. If a considered route is disconnected, then it is reconnected by creating a sub-path between the disconnected nodes. The reconnection is done using the same process for initial path generation. The sub creating the need for the sub-function to reconnect them. To prevent infinite loops, the number of attempts (variable called *attempts2*) to repair the routes is limited to a user-defined number (*max_attempt2*). Each route is also checked to determine if it contains a loop. Existing loops are then eliminated using the loop removal sub-function. Similar to the reconnection sub-function, the attempts of loop removal (*attempts*) have a maximum number of *max_attempt1*. The

conditional statement at the end of the algorithm generates a new path if the reroute is still invalid after the repair procedures have been carried out.

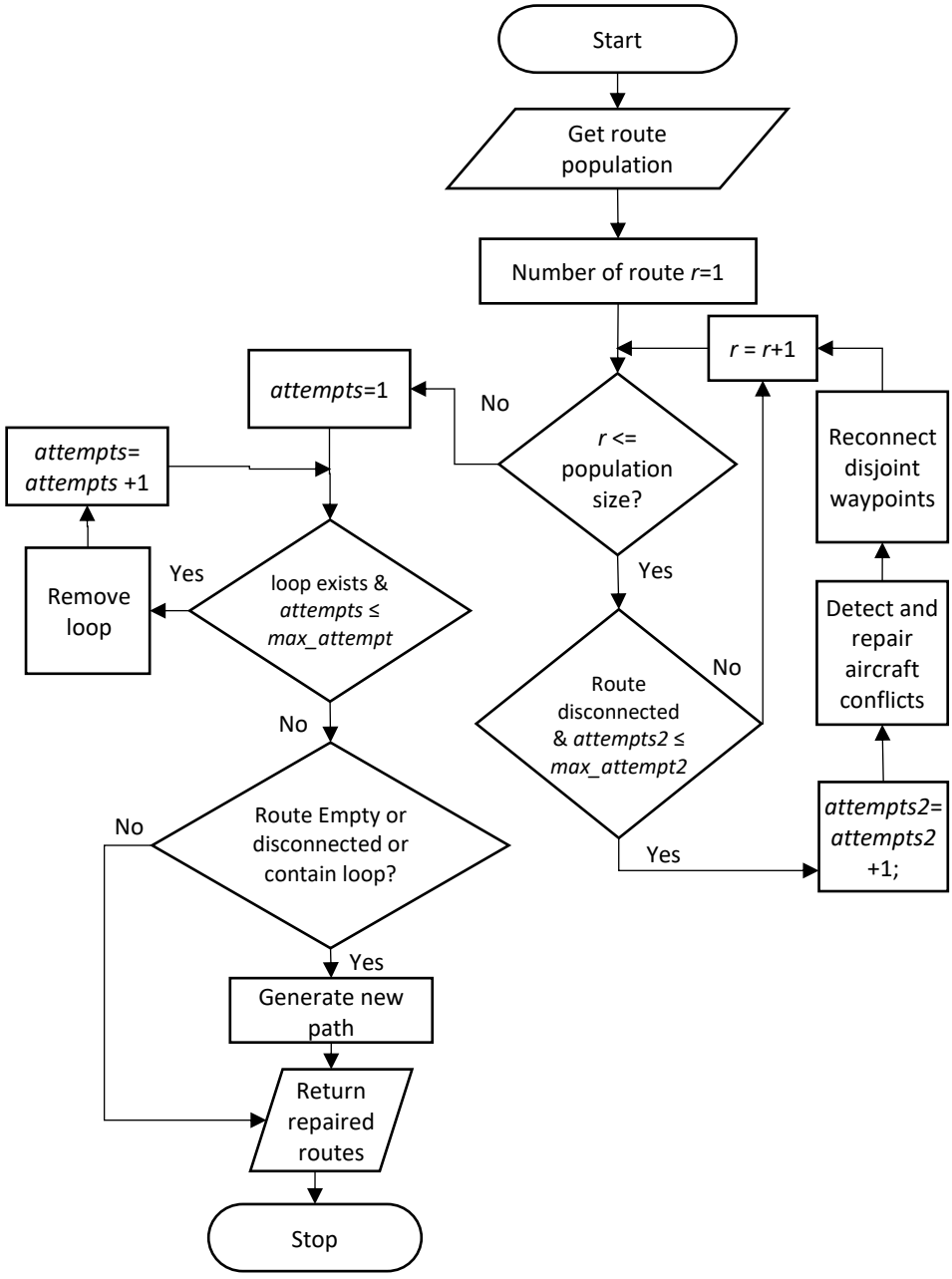


Fig. 5.12: Flowchart for repair function

The loop removal sub-function detects if there is a loop in a route by checking if a waypoint is repeated in the corresponding chromosome. When a waypoint is repeated, the waypoints between the repeated nodes are removed. The last occurrence of the repeating waypoint is also deleted. For GAAR02 (Ahn and Ramakrishna 2002), the repair function is made up of the loop removal function. Also, in GAAR02, the repair function is used after crossover and not after

mutation. This is because the mutation process in that case does not introduce disconnections or loops. For the proposed algorithm, the repair function is placed after mutation. This ensures that reroutes after mutation meet the constraints.

The reconnection sub-function is responsible for reconnecting disjoint routes. A route section between two waypoints is disjoint if there is no link between the two points. The sub-function attempts to build a new path between the two points using the algorithm for building initial paths.

The aircraft conflict detection routine checks that the path of the aircraft does not conflict with those of other aircraft. It checks that the times that the aircraft is to be at each waypoint does not coincide with the times other aircraft flying at the same level are going to be at the waypoint. The procedure is shown in Fig. 5.13.

```
Get trajectories of aircraft
For each waypoint in route of this aircraft
    Get estimated time of arrival at waypoint (t_w)
    If any other aircraft passes waypoint during time window
        Conflict is detected
    End If
    If conflict detected and waypoint is not start or destination
        Delete affected waypoint from route
        Remove waypoints from topological data
        Break
    End If
End For
Return route
```

Fig. 5.13: Algorithm for aircraft conflict detection

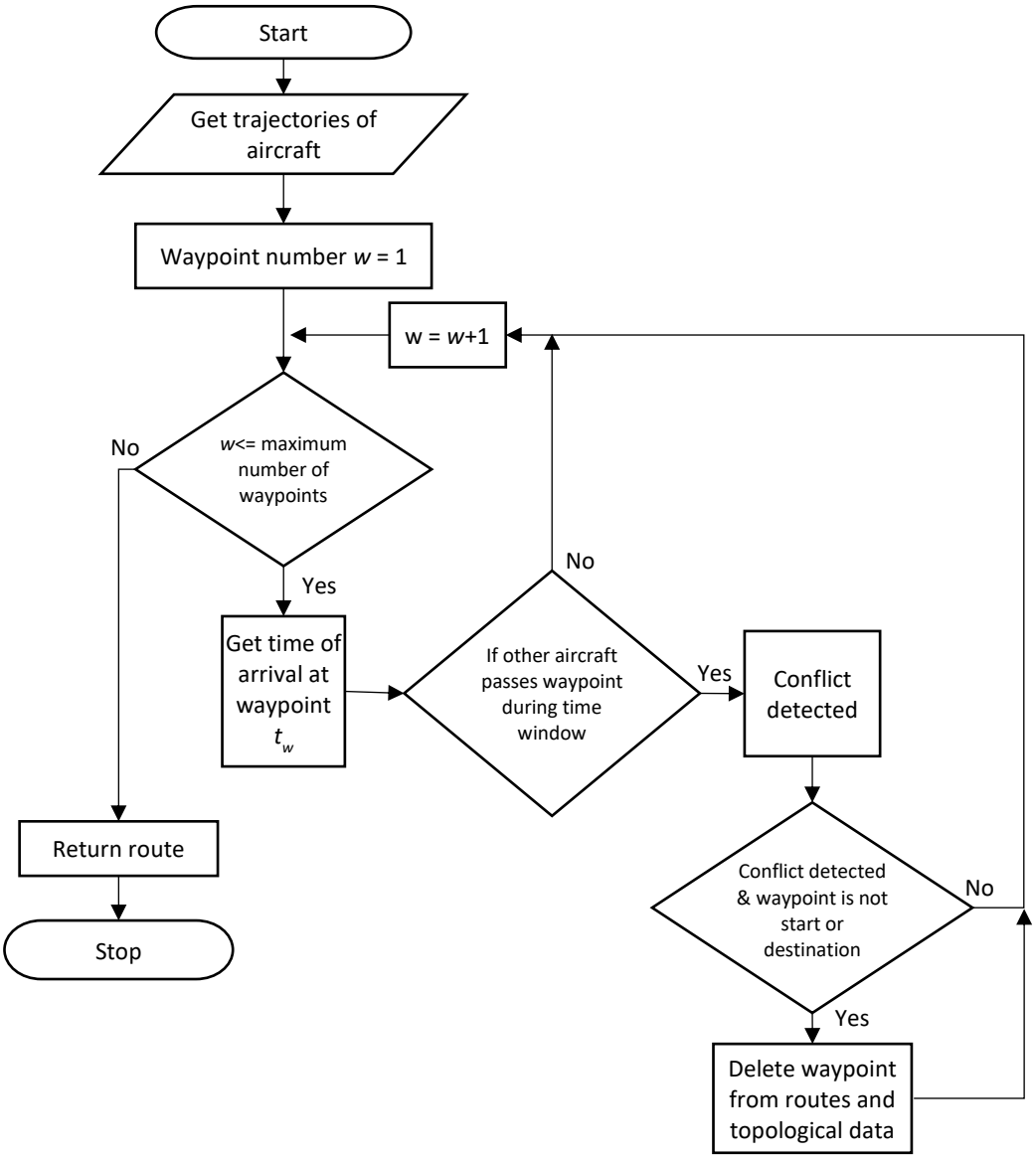


Fig. 5.14: Flowchart for aircraft conflict detection

The trajectories of surrounding aircraft are read by the algorithm. The algorithm checks for potential conflicts between these trajectories and its proposed reroute (Fig. 5.14). If the algorithm detects that there would be another aircraft at the intended waypoint (at the required altitude) during a time window. The lower and upper limits of the time window are given by $(t_w - t_{sep})$ and $(t_w + t_{sep})$ respectively. $t_w$ is the estimated time of arrival of the considered aircraft at the waypoint and $t_{sep}$ is the required time separation between aircraft. If a conflict is detected, a new path avoiding the waypoint is built. It is assumed that the considered aircraft would be the one to change its path, and that aircraft are able to communicate their planned paths. In addition, such modified paths could be submitted to air traffic control for approval.

### 5.3.9 Visualisation of results

This module is used for the graphical display of flight paths, and plots details such as location of weather regions. In addition, the costs associated with reroutes are displayed. These features help user interaction and decision-making.

### 5.3.10 Termination criterion

The rerouting algorithm is stopped when certain conditions are met. In this work, the termination criterion was number of generations. This was chosen so that the execution time of the algorithm can be customised, using the trade-off between optimality and the urgency of the rerouting.

### 5.3.11 Contribution of the proposed GA-based algorithm

As discussed in this sub-section, the proposed GA-based algorithm improves the search ability of the rerouting algorithm by modifying the mutation process. The mutation process was improved by combining an insertion and replacement operation on the candidate routes, as shown in the flowchart in Fig. 5.10. This was done to increase the diversity of the population and improve the chances that the genes for the optimal solution are present in the candidate routes.

## 5.4 A discrete firefly algorithm (DFA) approach to flight path reroute

Firefly algorithm (FA) mimics the behaviour of fireflies. Fireflies emit light flashes and are attracted to one another's light (Yang 2009). This light varies with distance and light absorption of the transmission medium. The value of the objective function at a firefly's location determines the light intensity of the firefly. It is assumed that the attractiveness of a firefly is dependent on its brightness. In the original FA, the distance between fireflies was the Cartesian distance between them. A firefly moves towards more attractive flies. FA has fewer parameters, is simpler to implement, and converges fast (Yang 2009).

A DFA-based technique is used for the rerouting of flight paths during adverse weather. The fireflies represent the candidate routes, similar to the chromosome structure defined earlier. The proposed approach (DFA17) is shown in Fig. 5.15 and Fig. 5.16 below.

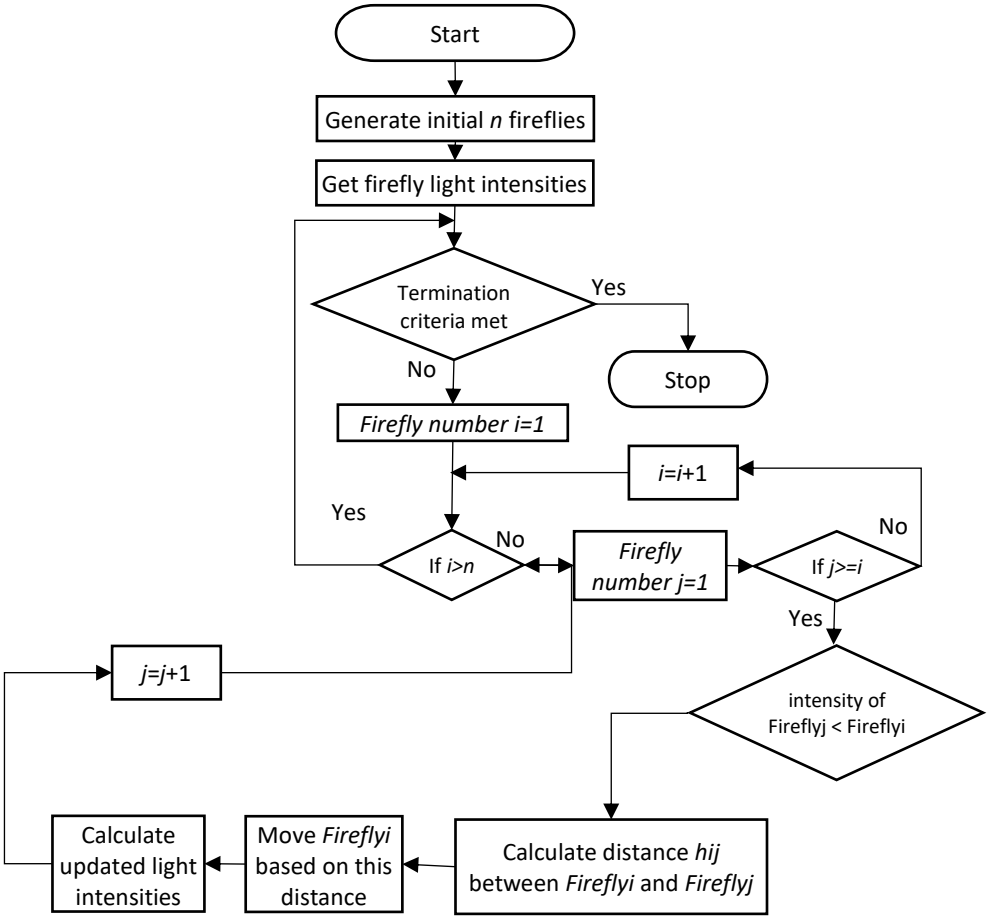


Fig. 5.15: Flowchart for DFA-based weather avoidance (DFA17)

```
Generate initial set of n fireflies
Get light intensities of the fireflies I_i
While (termination criteria not met)
        For Fireflyi = first firefly to n^th firefly
                For Fireflyj = first firefly to Fireflyi
                        If light intensity of Fireflyj < Fireflyi
                        Calculate distance h_ij between Fireflyi and Fireflyj
                        Move Fireflyi based on this distance
                        Calculate updated light intensities
                End For
        End For
        Get best firefly
End While
```

Fig. 5.16: DFA-based weather avoidance algorithm (DFA17)

The initial set of fireflies are generated using the procedure for obtaining the initial population for the GA-based algorithms defined earlier. The light intensity of a firefly is equivalent to the total costs incurred by the equivalent route, i.e.

$$I_i = J_t \tag{5.2}$$

During each iteration, the intensities of each firefly is calculated. Less bright fireflies move towards brighter fireflies. The distances between fireflies are equivalent to the Hamming distances $h_{ij}$ between them. This approach is similar to that used by (Osaba *et al.* 2017), who adapted the firefly algorithm for rich vehicle routing problem. The Hamming distance between two fireflies is given by the number of their constituent waypoints that do not correspond. In the present problem, the corresponding waypoints on a firefly must be located at the same position on the second firefly. The number of movements of a firefly towards a brighter firefly is given by an integer *m*. The number (Osaba *et al.* 2017) is randomly generated from the interval (2, $d_{ff}.\gamma^g$), that is:

$$m = Random\left(2\,,\ d_{ff}.\gamma^{g}\right) \tag{5.3}$$

where $d_{ff}$ = distance between the pair of fireflies, $g$ = iteration number, and $\gamma$ = light absorption parameter.

The movement of a firefly is done using an insertion and replacement process, similar to the mutation procedure for GAIM16. Copies of the firefly to be moved are made (*m* copies). A random position is located in each firefly of this set. A waypoint is selected randomly from the set of waypoints in the considered airspace. The selected waypoint is then inserted after the identified location in the firefly. After the insertion procedure, a waypoint in the firefly is randomly selected and replaced by a waypoint randomly chosen from the set of considered waypoints. The repair function is also run on the set of fireflies. The intensities of the modified fireflies are calculated and the best firefly is chosen to replace the original firefly. The termination criterion for the algorithm is the maximum number of generations.

### 5.4.1 Contributions for the DFA-based algorithm

The proposed DFA17 technique, shown in the flowchart of Fig. 5.16, uses a discrete approach to solution search, allowing the firefly algorithm to be applied

to network-based problems. The DFA17 technique improves the search for optimal routes by using a modified movement strategy. The movement strategy involves a combined replacement and insertion operation on candidate route solutions, similar to the mutation process for GAIM16. The movement strategy increases diversity and increases the chances that waypoints of the optimal route are present in candidate routes during the solution search. This has the overall effect of improving the search ability of the proposed algorithm.

## 5.5 Summary

This chapter has presented algorithms for improved generation of alternative flight path during adverse weather. The algorithms supported the use of the developed model that considered the effects of adverse weather reroutes on passengers, along with aircraft operational costs. To tackle the problem of premature convergence and entrapment in local optima, a GA with modified mutation procedure was used. In addition, a DFA-based algorithm for the path rerouting problem was presented.

# Chapter 6 : RESULTS AND ANALYSIS

## 6.1 Introduction

This chapter presents simulation results of the developed algorithms and models. The effects of parameters such as mutation rate, crossover rate, population size and number of generations were observed and discussed. Results for different path rerouting scenarios have also been presented and analysed.

## 6.2 Simulation environment

There was a lack of suitable open software for flight path routing research. Therefore, a software platform, called Flight Parout, had been developed to simulate and optimise flight paths. The proposed algorithms were implemented in the MATLAB programming language. The personal computer used for the simulations had the following settings: a Windows 8.1 operating system, 6 GB random access memory, a 3.20 GHz Intel Core i5-3470 processor and 500 GB hard drive storage. The software platform was designed as modulars to enable the improvement and testing of its constituent parts. Components of the platform are shown in Fig. 6.1 and discussed below.

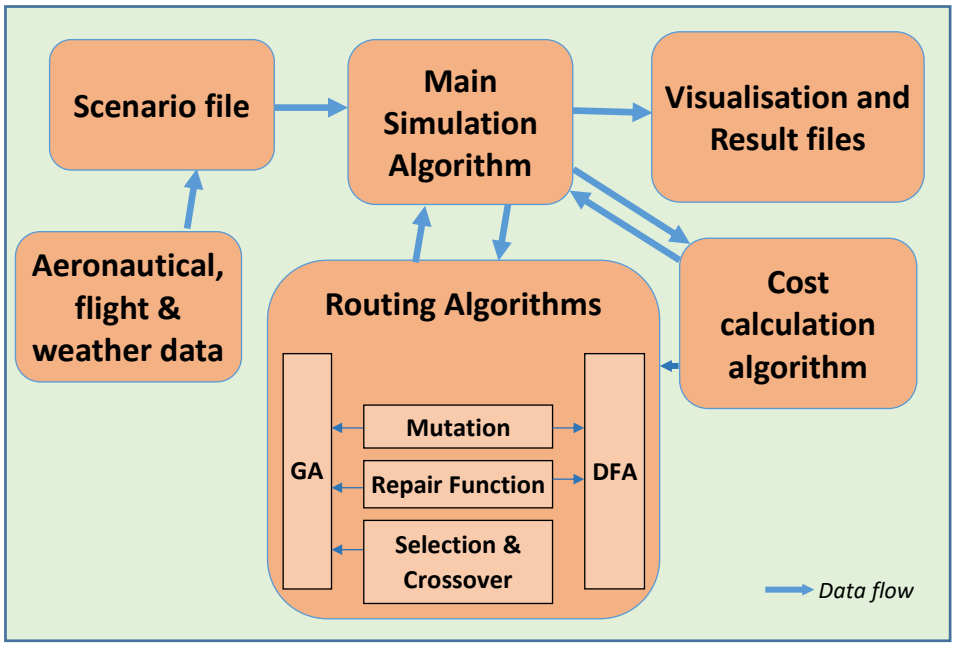


Fig. 6.1: Components of the simulation platform

The scenario file is used for setting up a simulation. It specifies the values of simulation parameters, such as population size and mutation probability. In addition, it specifies the type of path rerouting algorithms used for the simulations. A sample scenario file is shown in Fig. 6.2.

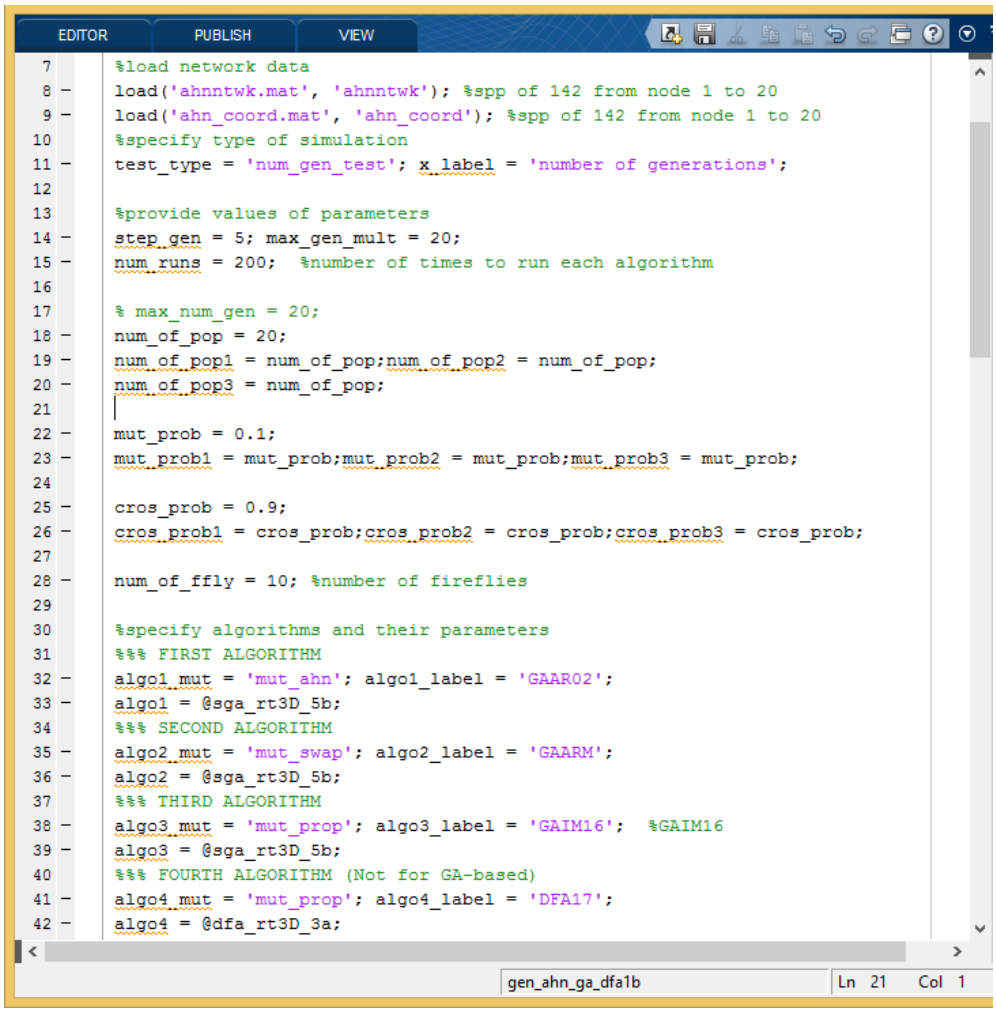


Fig. 6.2: Simulation scenario file

The main simulation algorithm is responsible for calling a rerouting algorithm. It passes inputs, such as flight, aeronautical and weather data, to the rerouting algorithm. The main simulation algorithm also handles and processes outputs from the algorithm, such as the calculation of average costs for the rerouting algorithms. The main simulation algorithm saves the output of the simulation in the result files. Visual display of path reroutes and the airspace is produced by the visualisation module. The components of the routing algorithms module include GA, DFA, cost calculation, mutation, selection, crossover and repair functions. The routing algorithms module receives data flow, including

configuration data, and passes them to its component elements. Result data from the rerouting module is also passed to the main simulation algorithm. The cost calculation algorithm evaluates the cost associated with a candidate solution. The algorithm uses the specified cost model to obtain these values.

## 6.3 Benchmark 1: Weighted Network Scenario

### 6.3.1 Introduction

In this case, an airspace with known waypoints was considered. The benchmark problem of Ahn and Ramakrishna (2002) was adopted for the layout of the network. The network graph is made up of waypoints (nodes) interconnected by weighted links as shown in Fig. 6.3 below. The weights of each link in the network represented the total costs associated with the link in US dollars per passenger, less any weather impact costs.

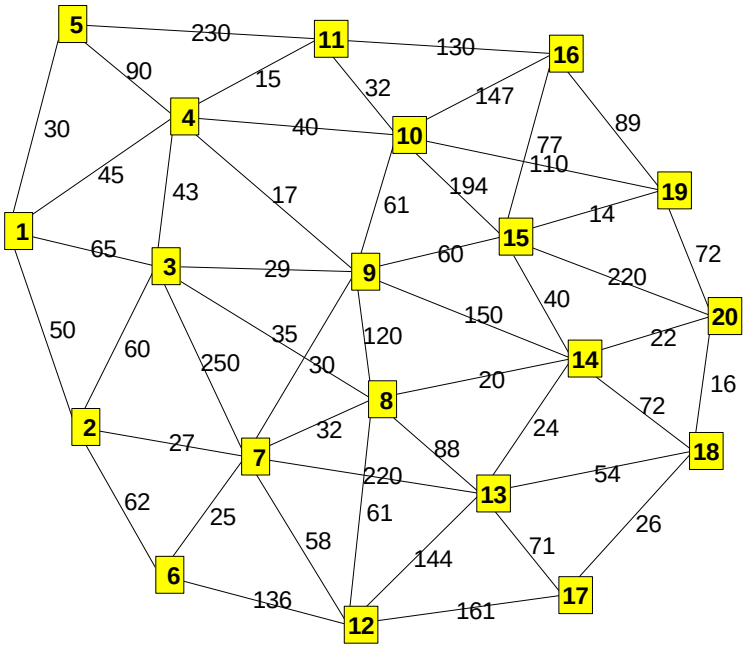


Fig. 6.3: Benchmark 1 network scenario (Ahn and Ramakrishna 2002)
(where yellow rectangles represent waypoints, black lines are route links, numbers on lines are link costs)

### 6.3.2 Effects of mutation rate (Benchmark 1)

In this scenario, the effects of mutation rate on the quality of solutions obtained by the algorithms was investigated for Benchmark 1. The source code for the

scenario is given in Appendix I. Three GA-based algorithms with different mutation strategies were used. The algorithms were discussed in the previous chapters and they were GAAR02, GAARM and GAIM16. GAAR02 was GA using the mutation technique of Ahn and Ramakrishna (2002). GAARM used a replacement strategy for mutation, in which a waypoint in a candidate solution is randomly replaced with another waypoint selected from the solution space. In addition to a replacement strategy, GAIM16 used both a replacement and an insertion strategy for the mutation procedure. The goal was to find the route with the least costs from waypoint 1 to waypoint 20. The simulation parameters are shown in Table 6.1. The mutation rate was varied from 0.05 to 0.5. The values of costs obtained by the algorithms for each mutation rate were recorded. The process was repeated 200 times and average values were calculated. The same initial population was used for each run.

Table 6.1: Simulation parameters for Scenario 1

| **Parameter** | **Value** |
|---|---|
| Crossover rate | 0.9 |
| Number of generations | 50 |
| Population size | 40 |

Fig. 6.4 shows the variation of average cost obtained as the value of mutation rate was changed. At a mutation rate of 0.1, an average cost of $142.00 was obtained by GAIM16. It is known that the cost of the shortest route was $142.00 (Ahn and Ramakrishna 2002). Hence, GAIM16 was able to obtain the optimal solution.

The average cost of paths obtained by GAARM at a mutation rate of 0.1 was $143.00. This value was worse than that obtained by GAIM16 and had a difference of $1.00 from the optimal solution. However, the highest cost was obtained by GAAR02, with an average cost of $169.91. This had a difference of $27.91 from the optimal value.

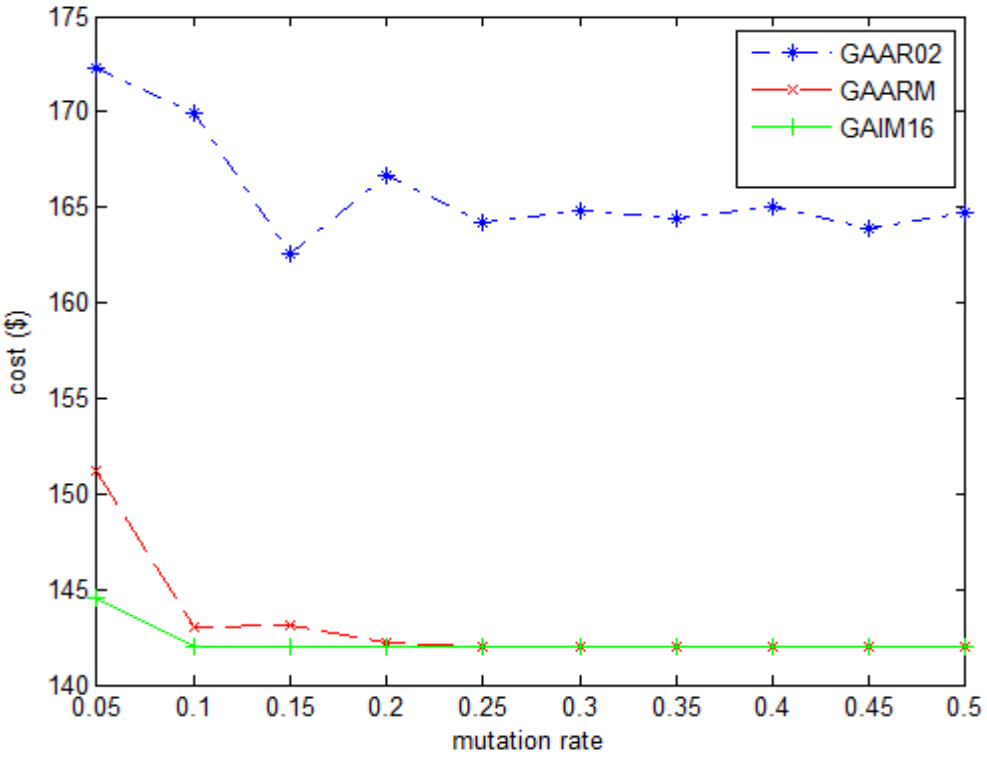


Fig. 6.4: Variation of costs with mutation rate

Table 6.2 shows further results of the simulation scenario. For the initial mutation rate of 0.05, the average cost obtained by GAAR02 was $172.25 and the standard deviation was $37.08. Routes obtained by GAARM had an average cost of $151.17 and a standard deviation of $23.13. GAIM16 had the best performance of the three algorithms. It obtained routes with average costs of $144.54 and standard deviation of $10.81.

It can be seen that, on the whole, higher levels of mutation led to lower average costs of routes. However, Fig. 6.4 shows that the change in mutation rate from 0.05 to 0.1 had the greatest impact on GAARM. The solution costs for GAARM decreased progressively till 0.2, where obtained solution costs became close to the optimal value.

The overall trend for GAAR02 shows that solution cost decreased with increase in mutation rate. However, there were exceptions at mutation rates of 0.2 and 0.5. The reason for this is mostly because of the probabilistic elements (such as initial population generation, mutation, and crossover functions) of the technique.

Mutation rates higher than 0.1 did not lead to much improvements for GAIM16. This was expected because it was able to achieve costs close to the best possible value (optimum cost of 142) for those mutation rates. This implies that GAIM16 had a suitable convergence to optimality for the benchmark problem for the mutation rates.

Table 6.2: Variation of costs at mutation rate of 0.05

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) | Probability of optimality |
|---|---|---|---|---|---|
| **GAAR02** | 172.25 | 37.08 | 142.00 | 279.00 | 0.4050 |
| **GAARM** | 151.17 | 23.13 | 142.00 | 267.00 | 0.8200 |
| **GAIM16** | 144.54 | 10.81 | 142.00 | 216.00 | 0.9400 |

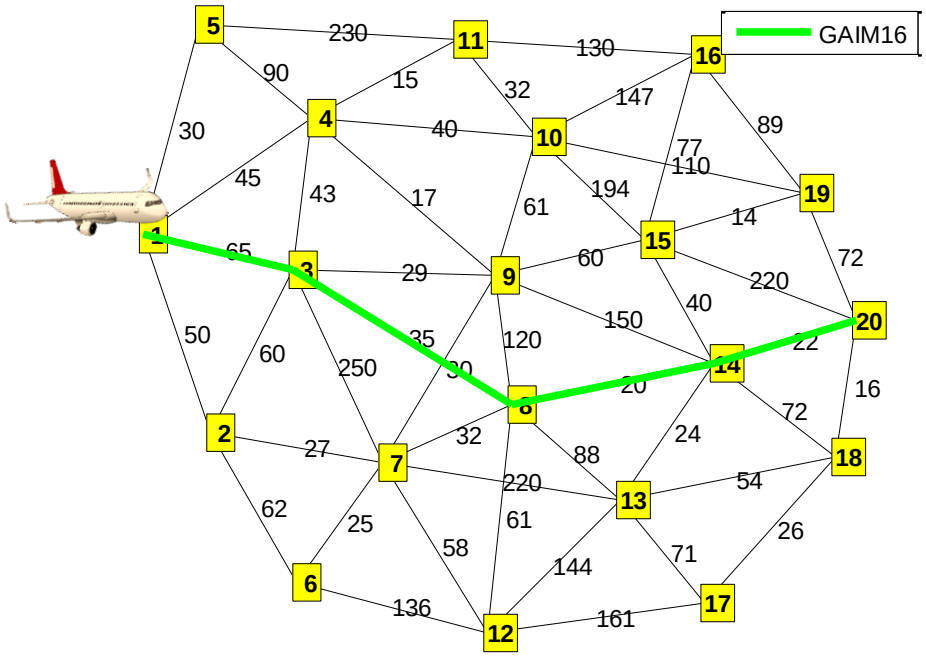


Fig. 6.5: Optimal path for Benchmark 1

(where yellow rectangles represent waypoints, black lines are route links, numbers on lines are link costs)

For a mutation rate of 0.05, the costs of longest paths obtained by GAAR02, GAARM and GAIM16 were $279.00, $267.00 and $216.00 respectively. All three algorithms obtained the optimal costs of $142.00. This corresponded to a route of 1-3-8-14-20, shown in Fig. 6.5. However, the probability of GAAR02 obtaining

the optimal path was only 0.4050 for a mutation rate of 0.05 (see Table 6.2). For GAARM, the corresponding probability was 0.8200. GAIM16 had the best probability of 0.9400 in obtaining the optimal path.

For a mutation rate of 0.5, the average cost of paths generated by GAAR02 improved slightly to $164.73 (see Table 6.3). A better value of $142.00 was got by GAARM, which was the optimal value. Similarly, GAIM16 had an average cost of $142.00. The standard deviation of the costs for GAAR02, GAARM and GAIM16 was $32.99, $0 and $0 respectively.

Table 6.3: Variation of costs at mutation rate of 0.50

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) | Probability of optimality |
|---|---|---|---|---|---|
| **GAAR02** | 164.73 | 32.99 | 142.00 | 267.00 | 0.5250 |
| **GAARM** | 142.00 | 0.00 | 142.00 | 142.00 | 1.0000 |
| **GAIM16** | 142.00 | 0.00 | 142.00 | 142.00 | 1.0000 |

The highest (worst) costs for a mutation rate of 0.5 were $267.00, $142.00 and $142.00 for GAAR02, GAARM and GAIM16 respectively. For the highest costs, lower values were preferred because the considered scenario was a minimisation problem. Hence, the best values were those obtained by GAARM and GAIM16. The probabilities of obtaining the optimal cost were 0.5250, 1 and 1 for GAAR02, GAARM and GAIM16 respectively.

As expected, increased mutation led to improved solutions obtained because the mutation increased route diversity. The increased diversity increased the chances that the genes necessary for efficient solutions were introduced into the candidate routes. However, very high mutation would lead to increased computation and processing. On the measures of average costs, highest cost and probability of optimality, GAIM16 had the best performance. This is likely due to the improved ability of its mutation strategy to search the solution space and escape local optima.

### 6.3.3 Variation of population size for Benchmark 1

In this scenario, the effect of the number of individuals (population size) was analysed on the quality of solutions. The process was run 200 times and average

values were calculated. The GA-based techniques were also compared with the DFA-based approach. Population size (also used for number of fireflies) was varied from 5 to 50 for both the GA- and DFA-based algorithms. For the DFA technique, $\gamma$ was 0.95 and the probability of firefly movement was 1.00, as given in Osaba *et al.* (2017). The simulation parameters are shown in Table 6.4.

Table 6.4: Simulation parameters effects of population size

| **Parameter** | **Value** |
|---|---|
| Mutation rate | 0.10 |
| Crossover rate | 0.90 |
| Number of generations | 20 |
| Light absorption parameter $\gamma$ | 0.95 |

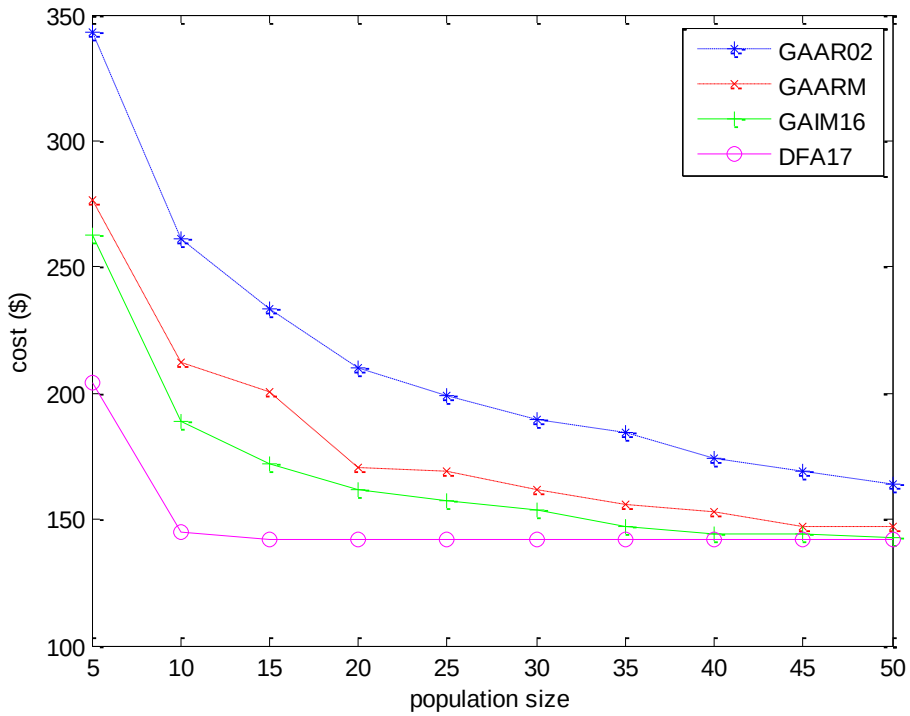


Fig. 6.6: Effects of population size for Benchmark 1

As shown in Fig. 6.6, results indicate that for a population size of 10, DFA17 was able to obtain solutions that were close to optimal values. At this point, the average costs of solutions obtained by DFA17 was $144.61, a difference of $2.61 from the optimal value. GAIM16 also performed sufficiently well and obtained

average costs of $189.14. The worst performance was by GAAR02 and the average costs it obtained was $261.33, a difference of $119.33 from the optimal solution. The average cost of $212.39 obtained by GAARM was slightly better than that of GAAR02.

From Fig. 6.6, it could be seen that the overall effect of increasing population size was the decrease in route costs. Hence, higher population sizes led to better solutions for all the considered algorithms. The largest change in costs occurred for a change in population size from 5 to 10 for all the four algorithms. For DFA17, population sizes greater than 10 did not lead to much improvements in average costs. This is because for population size of 10, the algorithm was already able to obtain costs close to the optimal value. The rest of the algorithms had progressively improved solutions as the population size was increased. For each population size, DFA17 had the lowest average costs. The second best solutions for all the population sizes were obtained by GAIM16, as can be seen in the figure. Whereas population size had a very noticeable effect on GAAR02, on the average, the costs of the routes it obtained were well above the optimal solution. It could be seen that the highest improvements in solutions for GAARM were for population sizes from 5 to 20. For population size of 50, GAARM had a route cost of $147.36 that was close to the optimal value and the solutions obtained by GAIM16 and DFA17.

Table 6.5 below shows more details of the simulation. For a population size of 5, the average of solutions obtained by GAAR02 was $343.41 with a standard deviation of $113.44. The cost of the worst solution was $715. GAARM, on the other hand, obtained a better average cost of $276.59, a difference of $134.59 from the optimal costs. The standard deviation of costs was $84.70.

Table 6.5: Variation of costs for population size of 5

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) | Probability of optimality |
|---|---|---|---|---|---|
| **GAAR02** | 343.41 | 113.44 | 142.00 | 715.00 | 0.0450 |
| **GAARM** | 276.59 | 84.70 | 142.00 | 487.00 | 0.0950 |
| **GAIM16** | 262.33 | 78.65 | 142.00 | 478.00 | 0.0750 |
| **DFA17** | 203.94 | 74.30 | 142.00 | 542.00 | 0.3800 |

The performance of GAIM16 at this point was the second best: it had an average cost of $262.33 and a standard deviation of $78.65. The best performance was by DFA17: the average of its solutions was $203.94, and the standard deviation was $74.30. In some simulation runs, the four algorithms obtained the optimal value. However, the probability of optimality for GAAR02 was the very low value of 0.0450, at the given population size of 5. The probabilities of optimality were slightly better at 0.0950 and 0.0750 for GAARM and GAIM16 respectively. DFA17 had the highest probability of optimality at 0.3800. This is attributed to its swarm-intelligence based search approach and the proposed movement technique.

For a population size of 25, the performance of all algorithms had markedly improved. The results are shown in Table 6.6. DFA17 was able to obtain the optimal cost of $142 in all the runs. The average cost of $157.20 obtained by GAIM16 was sufficiently close to the optimal value. GAARM obtained an average cost of $169.43. In this regard, the performance of the three algorithms was better than GAAR02. GAAR02 obtained an average cost of $199.04 and had a low 0.2850 probability of optimality. The corresponding probability for GAARM, GAIM16 and DFA17 were 0.5650, 0.6650 and 1.000 respectively. Similarly, the standard deviations were $53.67, $38.54, $28.08 and $0 for GAAR02, GAARM, GAIM16 and DFA17 respectively.

Table 6.6: Variation of costs for population size of 25

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) | Probability of optimality |
|---|---|---|---|---|---|
| **GAAR02** | 199.04 | 53.67 | 142.00 | 374.00 | 0.2850 |
| **GAARM** | 169.43 | 38.54 | 142.00 | 267.00 | 0.5650 |
| **GAIM16** | 157.20 | 28.08 | 142.00 | 259.00 | 0.6650 |
| **DFA17** | 142.00 | 0.00 | 142.00 | 142.00 | 1.0000 |

Table 6.7: Variation of costs for population size of 50

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) | Probability of optimality |
|---|---|---|---|---|---|
| **GAAR02** | 164.08 | 33.97 | 142.00 | 292.00 | 0.5700 |
| **GAARM** | 147.36 | 16.95 | 142.00 | 243.00 | 0.8800 |
| **GAIM16** | 142.78 | 5.52 | 142.00 | 198.00 | 0.9750 |
| **DFA17** | 142.00 | 0.00 | 142.00 | 142.00 | 1.0000 |

More performance improvements were obtained for a population size of 50, as shown in Table 6.7. GAAR02 had an average solution cost of $164.08 and a much-improved probability of optimality of 0.5700. The average costs for GAARM and GAIM16 were $147.36 and $142.78. The corresponding probability of optimality were 0.8800 and 0.9750 for GAARM and GAIM16 respectively.

It can be observed that as population size increased, the quality and optimality of solutions increased. This is explained by the increased diversity of properly generated populations. The initial populations for the algorithms were generated using a uniformly random process. However, increased population size could lead to increased processing and higher simulation time. DFA17 and GAIM16 produced adequate solutions even when presented with low initial population size. This is most likely due to the improved search ability when the proposed mutation and movement techniques are used.

### 6.3.4 Effect of number of generations

In this scenario, the number of generations is varied and the effects on costs were observed. The number of generations was varied from 5 to 50. The algorithms were run on Benchmark 1 for each value of the generation number. The process was repeated 200 times and average costs were calculated. Table 6.8 shows the simulation parameters used for the scenario.

Table 6.8: Simulation parameters for variation of number of generations' scenario

| **Parameter** | **Value** |
|---|---|
| Mutation rate | 0.10 |
| Crossover rate | 0.90 |
| Population size (GA) | 20 |
| Number of fireflies (DFA) | 10 |
| $\gamma$ | 0.95 |

Fig. 6.7 shows the variation of the costs for each algorithm in the scenario. The average cost of solutions decreased as number of generations increased. This expected because the considered problem is a minimisation problem. When the

number of generations was 20, GAAR02 obtained an average cost of $210.20. The results obtained by GAARM was much lower and was $178.64. The best value was obtained by DFA17: an average of $145.145. GAIM16 had the second best performance and obtained an average cost of $163.22.

From Fig. 6.7 it is seen that the average values obtained by GAIM16 were less than those of DFA17 up to the generations of 50. For greater number of generations, GAIM16 showed only minor improvements. For example, for number of generations of 90, the average costs obtained by GAIM16 and DFA17 was $142.37 and $143.61 respectively. The results indicate that the performance of GAIM16 was adequate. However, it needed more number of generations to converge to the optimal value, compared with DFA17.

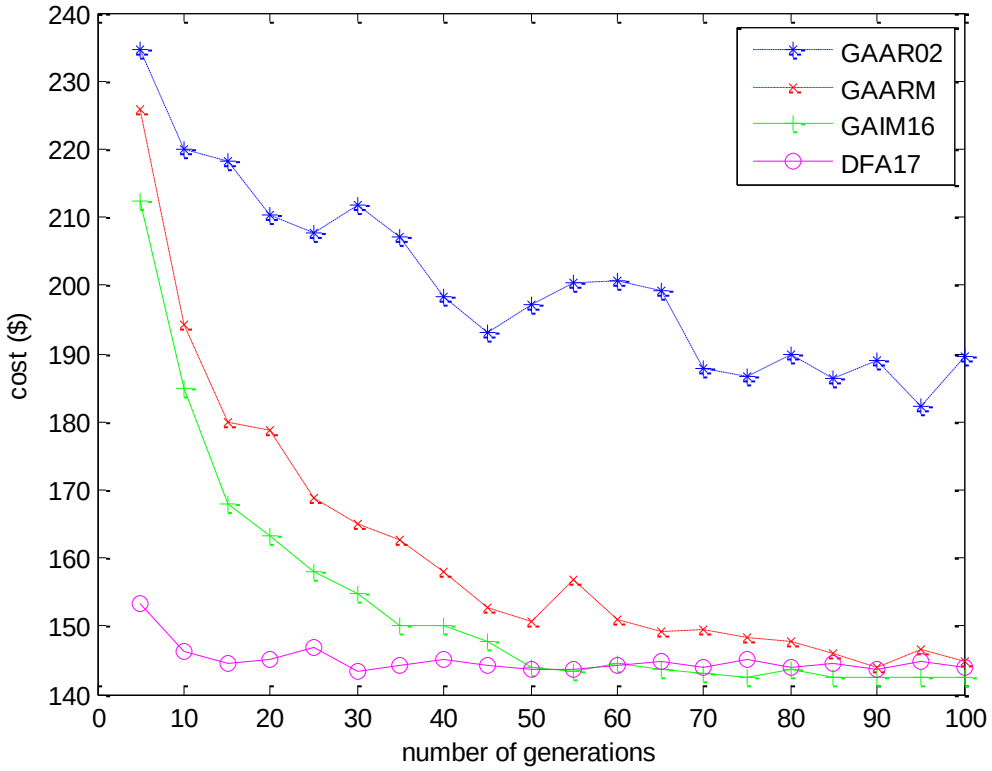


Fig. 6.7: Effects of number of generations for Benchmark 1

From Fig. 6.7, it can be seen that all algorithms experienced the largest decrease in costs when the number of generations was increased from 5 to 10. The costs obtained by GAIM16 improved considerably with each increase in number of generations until number of generations of 50. At this point, the algorithm had converged to the optimal solution. This decrease in solution costs as number of generations increased was also observed for GAARM. However, the solutions

obtained by GAARM were worse than the corresponding solutions for GAIM16. For number of generations greater than 5, DFA17 had solutions that were very close to the optimal value.

The overall trend for all the considered algorithms was a decrease in costs as number of generations was increased. This is expected because, in most cases, more number of generations means that more search operations were carried out. It could be observed that for all the considered algorithms, there were fluctuations and deviations from the decreasing trend. Examples include solutions obtained at number of generations of 55 and 95 for GAARM. The most algorithm with the most fluctuations was GAAR02. The reason for the deviation from the overall decreasing trend was likely because the considered algorithm had probabilistic operations such as mutations and initial population generation. It is expected that the fluctuations would be minimal as the number of algorithm runs is increased to a very large number.

Table 6.9 shows further results of the simulation. For number of generations equal to 5, an average cost of $153.23 was obtained by DFA17. This value was sufficiently close to the optimal value. The standard deviation of the DFA17 solutions was $22.51. The probability of obtaining the optimal value was 0.7000.

Table 6.9: Variation of costs for number of generations of 5

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) | Probability of optimality |
|---|---|---|---|---|---|
| **GAAR02** | 234.67 | 59.80 | 142.00 | 464.00 | 0.1200 |
| **GAARM** | 225.71 | 61.97 | 142.00 | 392.00 | 0.1700 |
| **GAIM16** | 212.34 | 55.30 | 142.00 | 374.00 | 0.2000 |
| **DFA17** | 153.23 | 22.51 | 142.00 | 240.00 | 0.7000 |

Table 6.10: Variation of costs for number of generations of 50

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) | Probability of optimality |
|---|---|---|---|---|---|
| **GAAR02** | 197.23 | 52.63 | 142.00 | 341.00 | 0.2750 |
| **GAARM** | 150.53 | 21.16 | 142.00 | 256.00 | 0.8050 |
| **GAIM16** | 144.06 | 8.76 | 142.00 | 208.00 | 0.9350 |
| **DFA17** | 143.58 | 10.95 | 142.00 | 240.00 | 0.9700 |

Table 6.10 showed the performance of the algorithms for a generation number of 50. All four algorithms showed improvements in performance. GAAR02 obtained an average cost of $197.23 with a standard deviation of $52.63. GAARM had an average cost of $150.53 with a standard deviation of $21.16. The second-best performance was that of GAIM16. It obtained an average of $144.06, a value that differs from the optimal value by only $2.06. DFA17 obtained an average cost of $143.58. All the algorithms were able to obtain the optimal value. However, the probability of achieving the optimal solution was only 0.2750 for GAAR02. The corresponding probabilities for the other algorithms were much better. The values were 0.8050, 0.9350 and 0.970 for GAARM, GAIM16 and DFA17 respectively. Code profiling results showed that the GAAR02, GAARM, GAIM16 and DFA17 algorithms had 225.20, 328.86, 383.92 and 1276.24 seconds of processing time respectively, over all the simulation runs. The timing results indicated that DFA17 took more processing time than GAIM16. For situations that require very fast response, GAIM16 might be used at the expense of less optimal results.

### 6.3.5 Effect of an adverse weather region

This scenario considers a situation in which there is a region of adverse weather along the default route. The adverse region was centred on waypoint 3 of Benchmark 1 and had a radius of 5 KM (see Fig. 6.8 below).

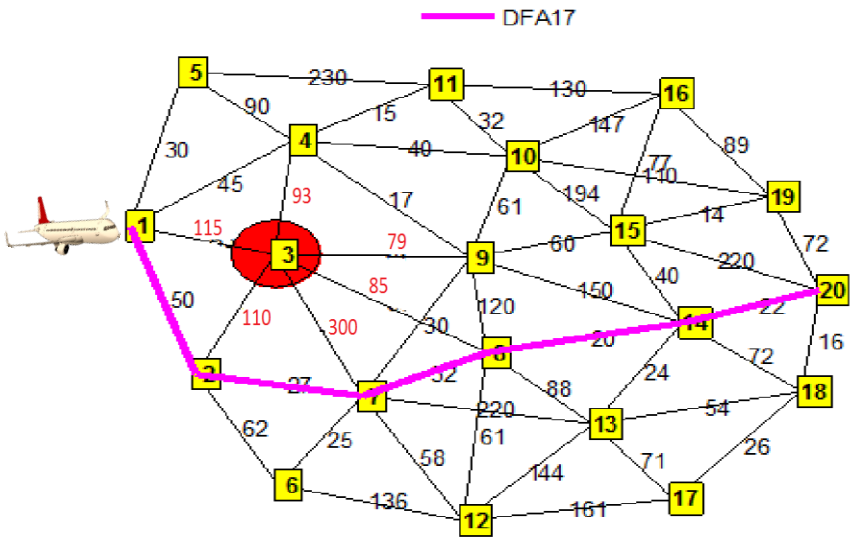


Fig. 6.8: Adverse weather avoidance for Benchmark 1

(where yellow rectangles represent waypoints, black lines are route links, numbers on lines are link costs, red circular region is adverse weather region)

The weather phenomena is considered to be a thunderstorm and had high intensity. The simulation parameters are the same as those of the previous scenario (Table 6.8).

Links that pass through the adverse weather region have an additional cost of $50 due to the weather phenomenon. These increase in cost is reflected by link costs labelled in red in Fig. 6.8.

As shown in Fig. 6.8, the rerouting module produced an alternative path that completely avoids the region of adverse weather. The best alternative path was 1-2-7-8-14-20. It could be seen that the path first deviates around the region of adverse weather before re-joining the original path (1-3-8-14-20). The total cost of the weather-avoiding path was $151.00. Other algorithms were able to obtain this path solution in some of the runs. However, from Table 6.11, GAAR02 had the lowest performance. For number of generations of 5, the cost of the worst solution it produced was as high as $447.00. The average cost of the solutions it produced was $281.54, with a standard deviation of $67.40. For GAARM, the average cost was better at $262.42 and standard deviation of $61.86. The second-best performance was that of GAIM16: the average cost of its reroutes was $257.47. The associated standard deviation was $53.22. This improved performance, compared with that of GAAR02, was most likely due to the enhanced search ability of the proposed mutation. The best performance, in terms of average cost, was $188.44 and was obtained by DFA17.

Table 6.11: Variation of costs for number of generations of 5 (single weather region scenario)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 281.54 | 67.40 | 151.00 | 447.00 |
| **GAARM** | 262.42 | 61.86 | 151.00 | 462.00 |
| **GAIM16** | 257.47 | 53.22 | 151.00 | 437.00 |
| **DFA17** | 188.44 | 37.04 | 151.00 | 292.00 |

Table 6.12: Variation of costs for number of generations of 25 (single weather region scenario)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 243.23 | 60.12 | 151.00 | 445.00 |
| **GAARM** | 211.82 | 42.10 | 151.00 | 310.00 |
| **GAIM16** | 191.28 | 37.06 | 151.00 | 292.00 |
| **DFA17** | 169.44 | 32.29 | 151.00 | 339.00 |

Table 6.13: Variation of costs for number of generations of 50 (single weather region scenario)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 234.52 | 51.42 | 151.00 | 409.00 |
| **GAARM** | 191.98 | 36.24 | 151.00 | 269.00 |
| **GAIM16** | 172.43 | 27.54 | 151.00 | 267.00 |
| **DFA17** | 166.10 | 29.19 | 151.00 | 267.00 |

Similar trends were observed at number of generations of 25 (Table 6.12). Further improvements in the quality of solutions were observed. The average cost for DFA was $169.44 and the corresponding standard deviation was $32.29. Similarly, the average cost generated by other algorithms improved to $243.23, $211.82 and $191.28 for GAAR02, GAARM and GAIM16 respectively. The costs of the worst paths generated by DFA17 increased from $292.00 to $339.00. This is most likely due to diversity in the population. The algorithm may initially produce worse candidate solutions while ensuring the solution space is properly searched. At number of generations of 50, its worst cost had dropped to $267.00 (Table 6.13). The average cost for DFA17 at this point was $166.10 and the standard deviation was $29.19.

The variation of cost with increased number of generations is shown in Fig. 6.9. The results indicate that GAIM16 obtained costs that were lower than those of GAAR02 and GAARM, at all the considered generations. Similarly, DFA17 obtained costs that were lower than those of GAIM16 over all the considered generations. The developed algorithms were, therefore, able to produce paths that more efficiently avoided the adverse weather region.

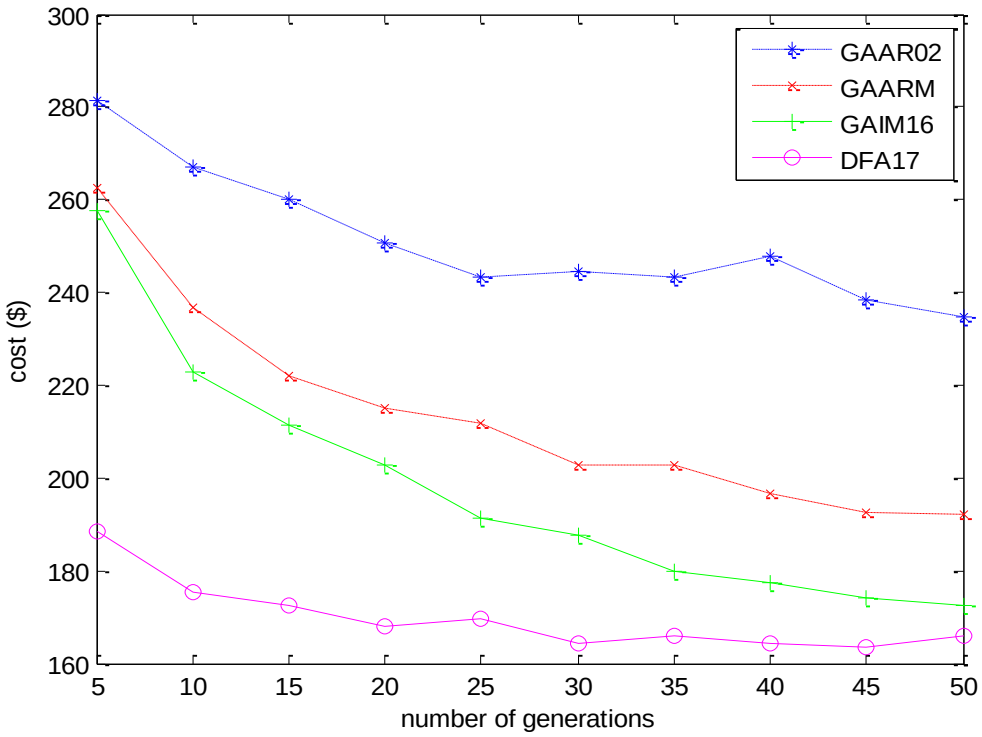


Fig. 6.9: Variation of cost for adverse weather avoidance

Similar to the case without weather region, the overall effect of increasing number of generations was a decrease in costs. For all the algorithms, the greatest change in cost occurred as number of generations increased from 5 to 10. From Fig. 6.9, it could be seen that for GAAR02, costs steadily decreased with each of the increments in number of generations from 5 to 25. There was less change in costs for number of generations of 25 to 40. There was a slight increase in costs at number of generations of 30 and 40. This is attributed to the probabilistic nature of the algorithm.

For each number of generation, the improvements of costs obtained by GAARM over those of GAAR02 were large. The costs obtained by GAARM steadily decreased as the number of generations were increased. This is due to the additional search operations from the added generations. Similar behaviour was observed for GAIM16, except that the rate of cost decrease was higher than that of GAARM. This implies improved search ability due to the higher rate of convergence. This is especially so considering the lower costs obtained by GAIM16 for all the number of generations, compared with GAARM. Whereas the lowest costs was obtained by DFA17 for all the number of generations, it could

be seen that after number of generations of 10, the improvements in costs with each successive generation number was lower. This was likely because the obtained costs were already close to the optimal value.

### 6.3.6 Effect of multiple adverse weather regions for Benchmark 1

In this case, the effect of multiple weather regions of different intensities was considered. As shown in Fig. 6.10, two weather regions were located along the default flight path. The weather regions of high and low intensities were centred on waypoints 3 and 13 respectively. In addition, a region of adverse weather was located elsewhere at 7. The aim of the scenario was to derive an alternative path with minimal costs. The parameters for simulation are given in Table 6.8.

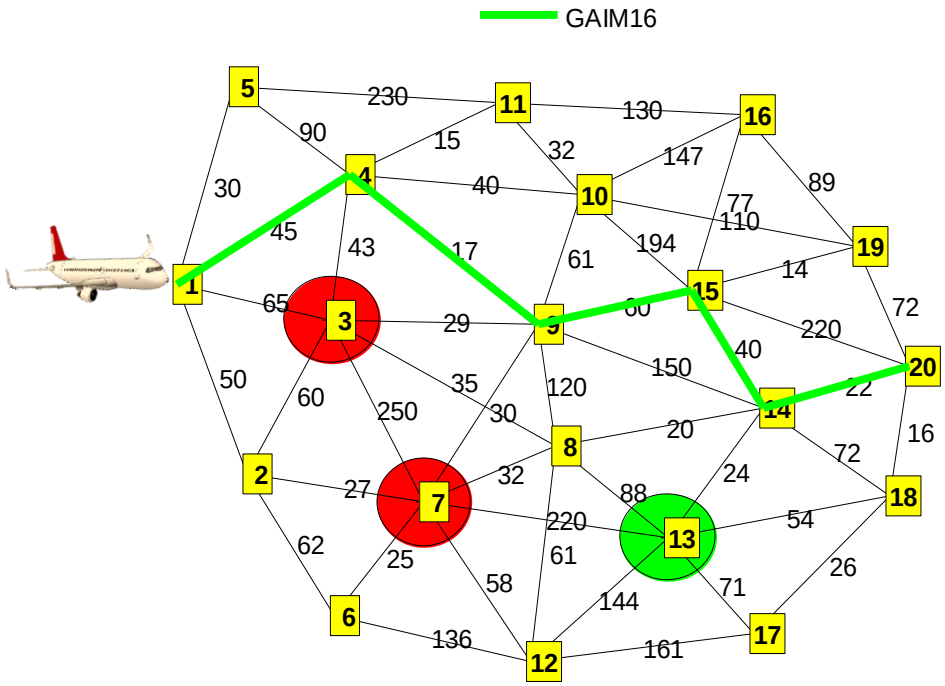


Fig. 6.10: Alternative route generation for case with multiple weather regions (Benchmark 1)
(where yellow rectangles represent waypoints, black lines are route links, numbers on lines are link costs, red and green circular shapes represent weather regions of high and low intensities respectively)

Fig. 6-10 shows the alternative path generated by GAIM16. The path was 1-4-9-15-14-20. The path avoided all three weather regions. The cost associated with the path was $184.00. This was an increase of $33.00, compared with $151.00 for the previous scenario with only one weather region. This is because the path had more regions to avoid, which lead to further deviation from the default path

of 1-3-8-14-20. The other three algorithms obtained the same alternative path solution.

From Fig. 6.11, it can be seen that, on the whole, cost obtained for the scenario decreased as number of generations was increased. The largest rate of improvement in costs for each increase in generation number was found in the range of 5 to 20 generations, for all the algorithms. Whereas GAAR02 had the worst cost, its rate of improvement in cost was low after number of generations of 20. This is in contrast with the performance of GAIM16. GAIM16 steadily improved its performance with increase in number of generations, with a relatively large improvement from generation number of 35 to 40. This brought the resulting solution close to that obtained by the best performing algorithm (DFA17). From its performance, it can be deduced that GAIM16 has improved search ability compared with GAAR02. This ensured that GAIM16 had better ability to escape from local optima in the solution space. GAARM progressively improved with increase in number of generations. However, it was unable to produce results that were better than those of GAIM16 and DFA17. This implies that GAAR02 had weaker search capability compared with GAIM16 and DFA17.

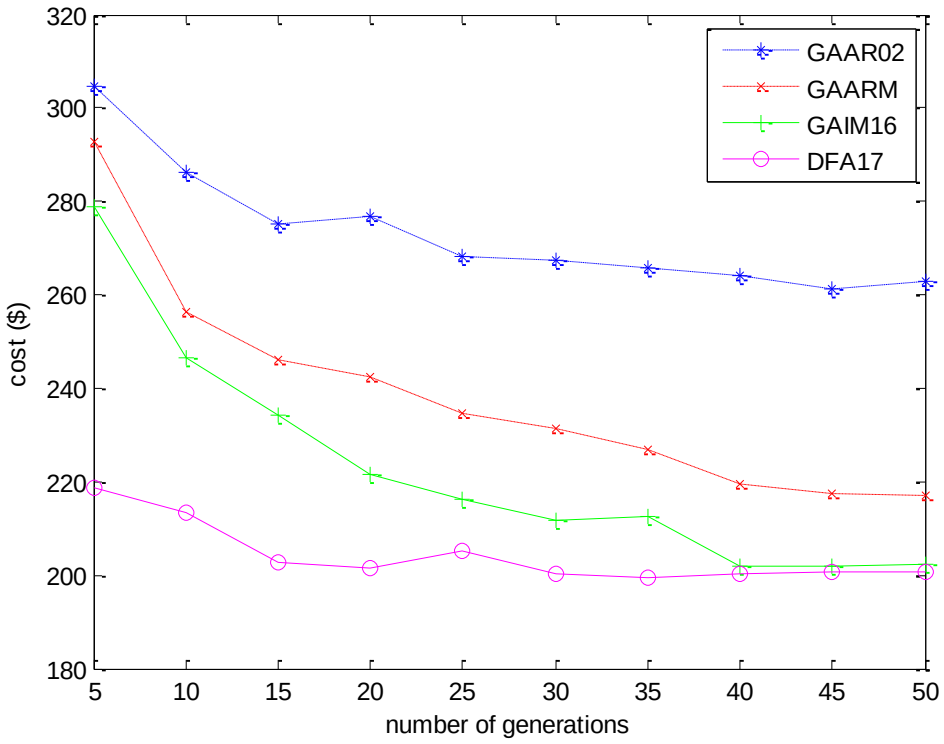


Fig. 6.11: Variation of cost with number of generations for multiple weather regions (Benchmark 1)

Table 6.14: Variation of costs for number of generations of 5 (for multiple weather regions)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 304.75 | 68.85 | 184.00 | 549.00 |
| **GAARM** | 292.69 | 58.54 | 184.00 | 451.00 |
| **GAIM16** | 278.93 | 52.84 | 184.00 | 474.00 |
| **DFA17** | 218.84 | 29.98 | 184.00 | 305.00 |

Similarly, the average costs increased with the increase in number of weather regions. For a generation number of 5, the average costs obtained was $304.75, $292.69, $278.93 and $218.84 for GAAR02, GAARM, GAIM16 and DFA17 respectively. This can be seen in Table 6.14. The corresponding standard deviation was $68.85, $58.54, $52.84 and $29.98. The results indicated that the best performance in terms of average costs and standard deviation was by DFA17. The second-best performance was by GAIM16.

Table 6.15: Variation of costs for number of generations of 25 (for multiple weather regions)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 268.03 | 58.41 | 184.00 | 497.00 |
| **GAARM** | 234.84 | 35.27 | 184.00 | 318.00 |
| **GAIM16** | 216.29 | 32.10 | 184.00 | 292.00 |
| **DFA17** | 205.34 | 25.76 | 184.00 | 342.00 |

Table 6.16: Variation of costs for number of generations of 50 (for multiple weather regions)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 262.98 | 50.75 | 184.00 | 423.00 |
| **GAARM** | 217.17 | 32.59 | 184.00 | 292.00 |
| **GAIM16** | 202.46 | 26.24 | 184.00 | 292.00 |
| **DFA17** | 200.83 | 24.89 | 184.00 | 300.00 |

The quality of solutions obtained by the algorithms were improved as the number of generations was increased (Fig. 6.11). Further details are shown in Tables 6.15 and 6.16. At number of generations of 50, GAAR02 obtained solutions that had an improved average cost of $262.98. GAARM and GAIM16 had average costs of $217.17 and $202.46, along with a standard deviation of $32.59 and $26.24 respectively. DFA17 had the best average cost of $200.83, and a standard deviation of $24.89.

## 6.4 Weather avoidance using grids

In this scenario, the airspace was divided into equally spaced sections and represented by a two-dimensional grid (Gleich 2008) at constant altitude. The gird was made up of waypoints in a ten by ten arrangement. Adjacent waypoints are connected to each other. Waypoints that are diagonally opposite to each other are also connected (Wang and Yang 2013).

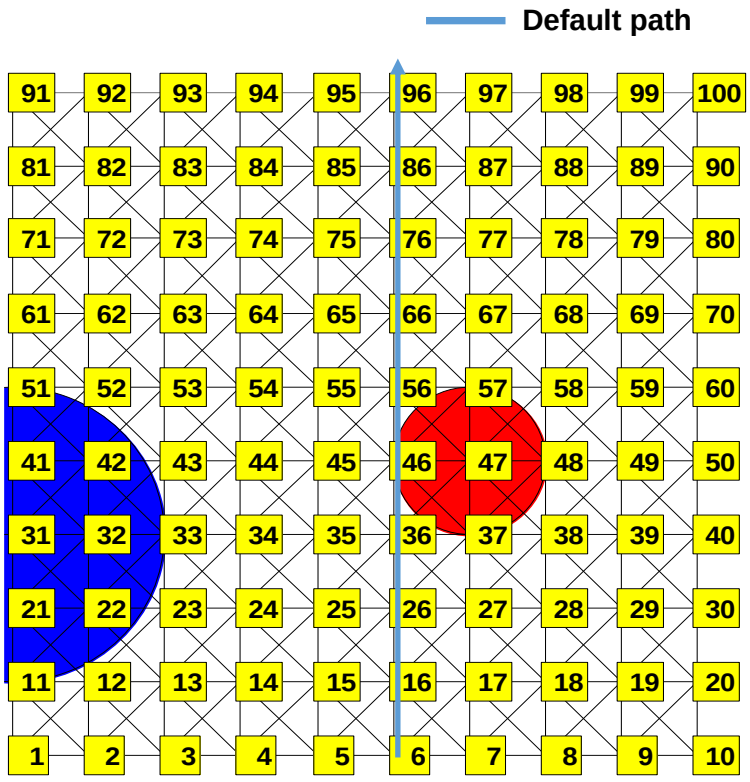


Fig. 6.12: Adverse weather regions for grid-based scenario

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively)

Fig. 6.12 shows the grid and areas with various degrees of weather impact. The distance cost of each horizontal or vertical link that connect adjacent waypoints is equal to the grid interval of 100 km. The distance cost of each link between diagonally opposite waypoints (such as between waypoints 1 and 12) is 141 km. This calculated by using trigonometric ratios and noting that the diagonal is the hypotenuse of a square shape with length of each side of 100 km. The red and blue regions show areas with high and medium levels of weather impact respectively. The weather region on the left (blue area) had a diameter of 200 km. The weather region towards the middle of the figure had a diameter of 100 km.

### 6.4.1 Effects of multiple weather regions for grid-based scenario

In this scenario, the default path for the aircraft considered was 6-16-26-36-46-56-66-76-86-96, which is a straight path from waypoint 6 to waypoint 96. The aim of the scenario was to find an alternative path between these waypoints. Such a path should have minimal cost for the adverse weather case. The effects of number of generations on obtained total cost was investigated. The weight of each component cost of flight delay, weather impact, missed connections was taken to be 1. The weight for fuel burn was assigned a value of 57 to penalise high fuel consumption. The weight for level change was assumed to be zero, because no flight level change was allowed for this scenario. The grid spacing was 100KM to reduce the amount of processing required. Smaller grid spacing might improve efficiency of rerouting but would incur greater amounts of computations and processing time in most cases. The aircraft was considered to have an average speed of 1000KM/hr with respect to the ground (Arıkan *et al.* 2016). The start time was assumed to be 10:00 GMT and end time was 10:54 GMT. The departure time for the connecting flight was 11:24 GMT.

For GAAR02 and DFA17 (Fig. 6.13), it can be observed that the improvement in solutions is reduced for number of generations above 15. For GAARM and GAIM16 marked improvements in obtained solutions could be seen up to number of generations of 20. The overall trend for all the algorithms was that increase in the number of generations led to solutions with improved (lower) costs. At number of generations of 20 and 25, DFA17 showed minor increases in cost. This is likely due to the randomness of the initial population generation function and repair

algorithm. It is expected that the plot for DFA17 would smoothen out as the number of algorithm runs is increased. However, increasing the number of runs also increases the analysis time for the algorithm. For most of the number of generations (up to 20), DFA17 had the best performance, compared with GAAR02, GAARM and GAIM16.

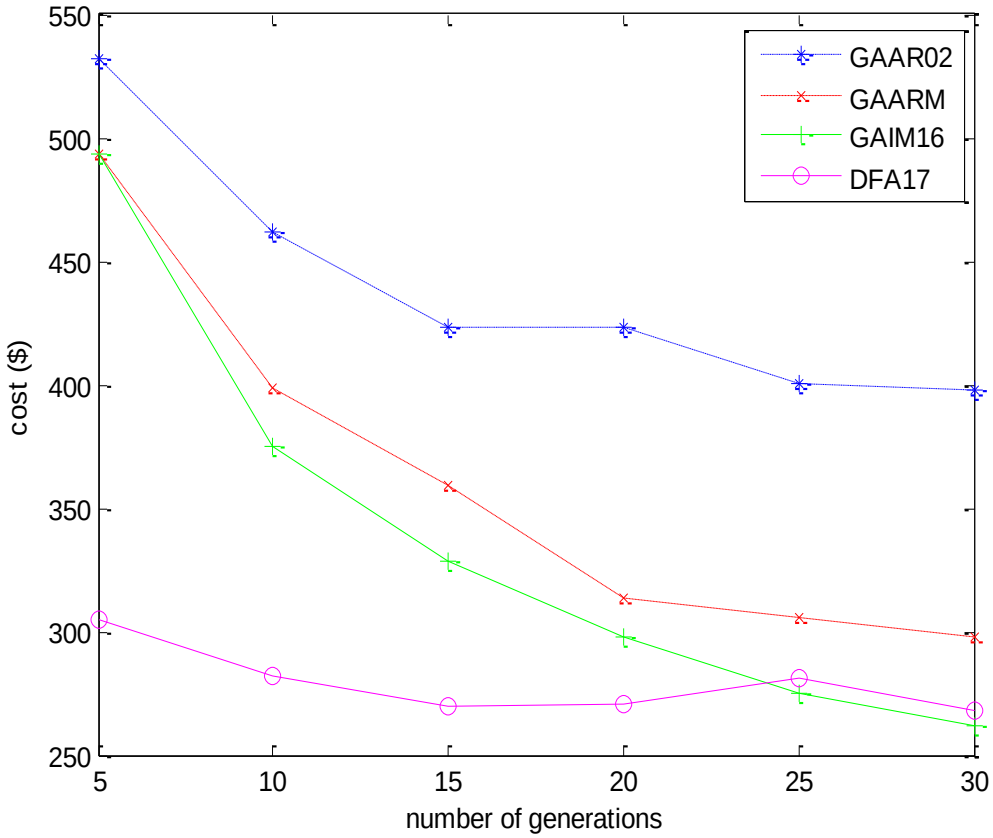


Fig. 6.13: Variation of cost for grid-based scenario with multiple weather regions

The lowest performance was by GAAR02, as shown by its plot that was above those of the other algorithms. GAARM had a better performance as indicated by lower costs at every number of generations, compared with GAAR02. The performance of GAIM16 was better than that of GAARM. GAARM and GAIM16 obtained similar costs at generation size of 5. However, for higher number of generations, GAIM16 produced solutions that had lower costs on the average. This implies that GAIM16 has better ability to search the solution space, compared with GAARM.

Table 6.17: Variation of costs for number of generations of 5

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 532.45 | 211.59 | 284.71 | 975.81 |
| **GAARM** | 493.32 | 168.71 | 226.03 | 975.81 |
| **GAIM16** | 493.10 | 181.30 | 226.03 | 975.81 |
| **DFA17** | 304.61 | 76.39 | 167.36 | 550.84 |

The performance for GAARM and GAIM16 was better with average costs of $493.32 and $493.11 respectively (Table 6.17). The average cost of paths derived by DFA17 at this point was the best by a far margin with a value of $304.61. However, Fig. 6.13 shows that for generations of 25 and 30, the average costs obtained by GAIM16 was slightly better than that of DFA17. This indicates that GAIM16 produced satisfactory solutions for the scenario under consideration. However, GAIM16 converged slower to the optimal value, compared with DFA17.

Table 6.18: Variation of costs for number of generations of 30

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 397.46 | 119.16 | 167.36 | 951.51 |
| **GAARM** | 298.23 | 103.45 | 108.68 | 751.18 |
| **GAIM16** | 261.61 | 81.09 | 108.68 | 609.52 |
| **DFA17** | 267.87 | 61.66 | 108.68 | 467.36 |

Table 6.18 shows further details of results at the number of generations of 30. The average cost of $261.61 obtained by GAIM16 was slightly better than of DFA17 ($267.87). However, both obtained a minimal cost of $108.68. DFA17 also had a lower standard deviation of $61.66, compared with $81.09. The cost of the worst solution produced by DFA was also better than that of GAIM16. This shows that for these metrics, DFA17 had better solutions than GAIM16. In addition, the average costs for both algorithms were better than the corresponding values of $397.46 and $298.23 obtained by GAAR02 and GAARM.

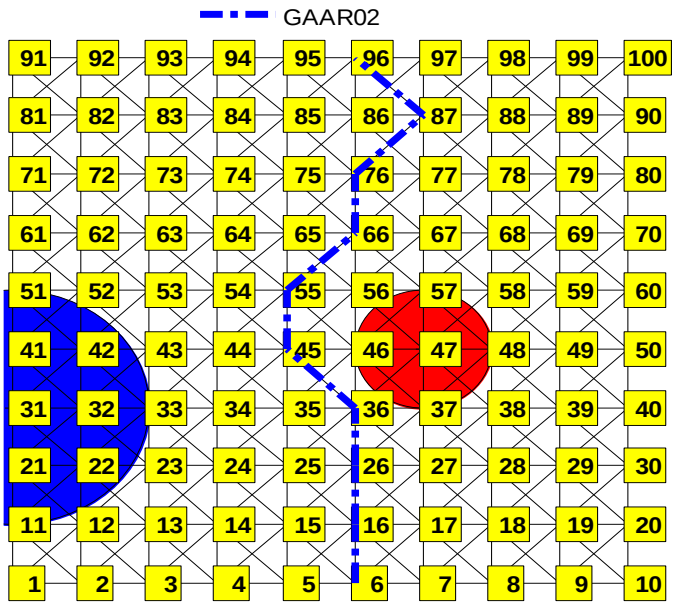


Fig. 6.14: Alternative paths generated for grid-based scenario by GAAR02 (where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities)

As shown in Fig. 6.14, the alternative path generated by GAAR02 was 6-16-26-36-45-55-66-76-87-96. This path followed the default route until it was close to the region of adverse weather located near the centre of the figure. It then avoided the weather region and reconnected to the default path at waypoint 66. The path went on to waypoint 87 before ending at the destination waypoint of 96.

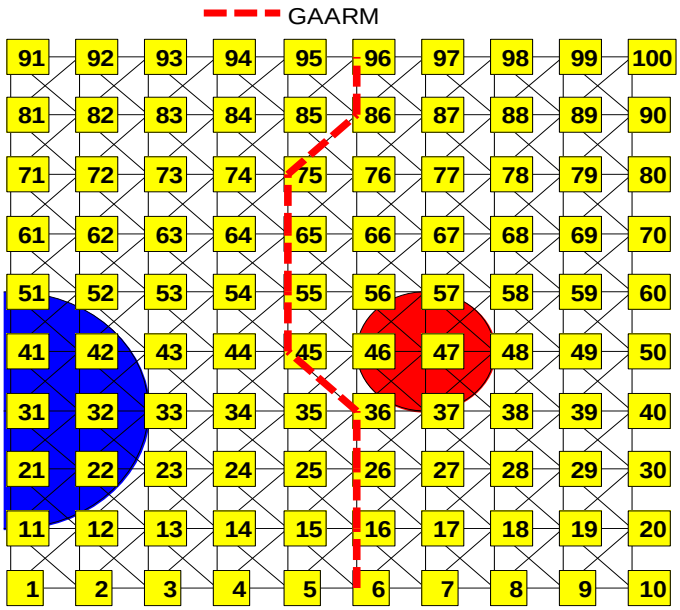


Fig. 6.15: Alternative paths generated for grid-based scenario by GAARM (where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities)

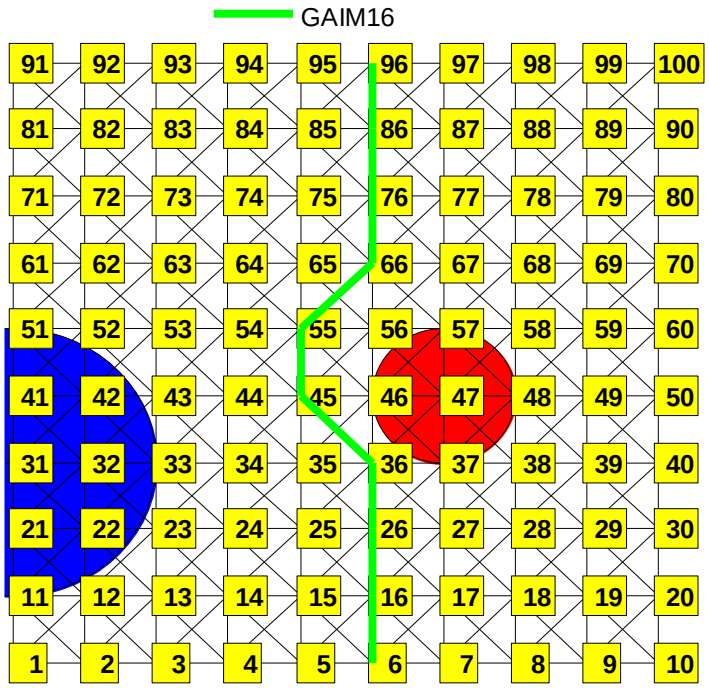


Fig. 6.16: Alternative paths generated for grid-based scenario by GAIM16

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively)

Whereas the same best cost of $108.68 was obtained by GAARM, GAIM16 and DFA17, there were some differences in the actual generated paths. Those generated by GAARM and GAIM16 were 6-16-26-36-45-55-65-75-86-96 and 6-16-26-36-45-55-66-76-86-96 respectively. The reroutes are shown in Figs. 6.15 and 6.16. These paths differ from each other at the points they deviate from and reconnect with the default path. The alternative path obtained by GAARM initially started on the default path. The alternative path deviated from the default one at waypoint 36 where it was close to the weather region located on the right. The alternative path did not re-join the default path until waypoint 86, which was located close to the destination waypoint. The reroute generated by DFA17 was 6-15-25-35-45-56-66-76-86-96 (Fig. 6.17). The path pre-emptively started the diversion process from the start waypoint of 6. It avoided the adverse weather region before re-joining the default path at waypoint 56.

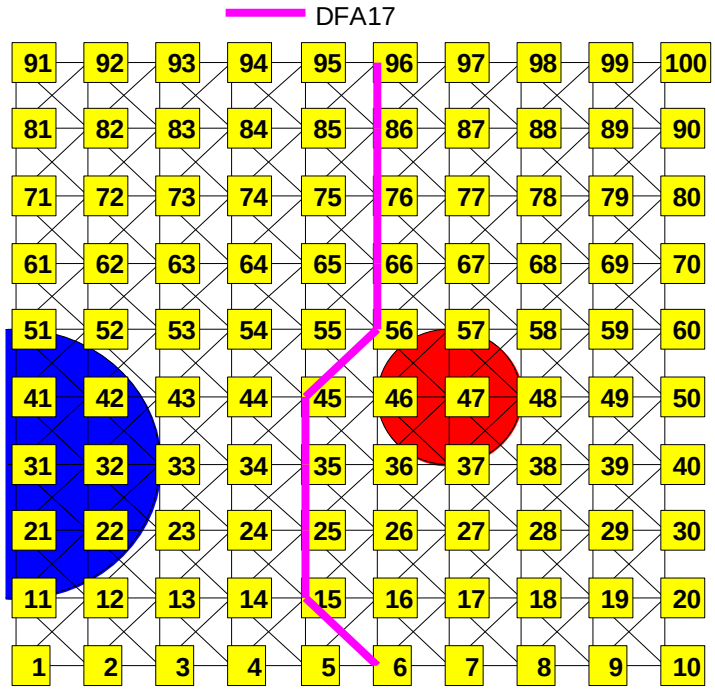


Fig. 6.17: Alternative paths generated for grid-based scenario by DFA17

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively)

The path generated by GAIM16 followed the default route until it encountered the weather region (Fig. 6.16) and then carried out a deviation around the weather region. It returned to the default path shortly after leaving the weather region. Among the paths produced by the four algorithms, this path is the preferable in most cases. This is because it produced the least deviation from the original path and associated disturbance to usual air traffic flow (Taylor *et al.* 2017a).

### 6.4.2 Scenario considering aircraft avoidance

In this scenario, in addition to the multiple weather regions, the effect of having another aircraft in the considered airspace was considered. The simulations considered the generation of alternative paths that avoids the weather regions while avoiding conflicts with another aircraft. The aircraft under consideration had a straight default path from waypoint 6 to waypoint 96. One other aircraft was in the area under consideration and had a path of 6-15-25-34-43, as shown in Fig.

6.18. Simulation parameters were the same as that of the previous section (Section 6.4.1).

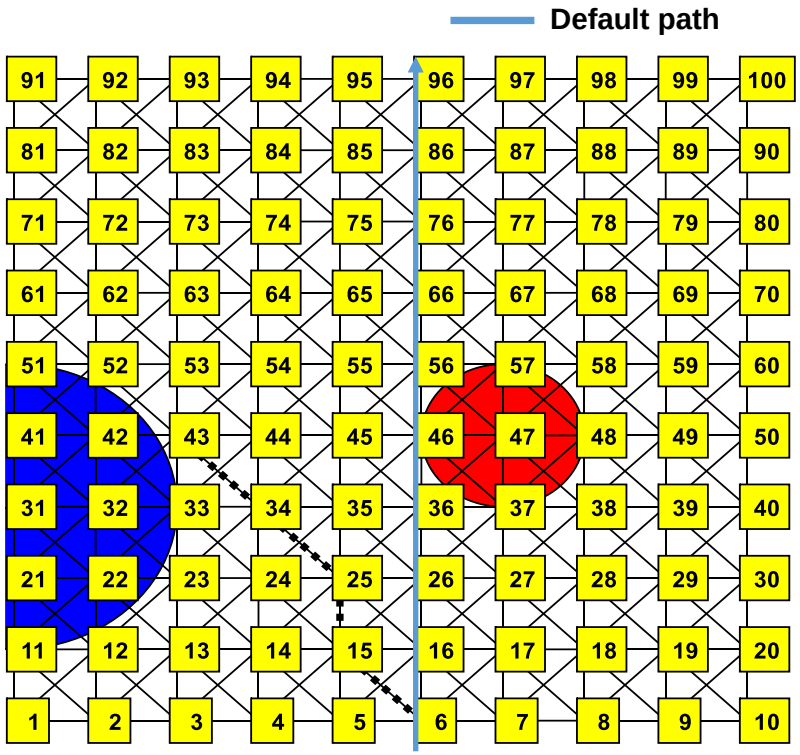


Fig. 6.18: Grid-based scenario with aircraft avoidance

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted line is path of other aircraft)

Fig. 6.19 shows the convergence of the cost obtained by the considered algorithms towards the minimum cost. GAAR02 had an average cost of $613.15 for number of generation of 5. This improves to $587.31 at generation number of 15, but not much improvement was made thereafter. This indicates that GAAR02 was unable to adequately escape local optima. Comparatively, GAARM had a lower average cost of $580.12 at number of generations of 5. It then progressively improved to $462.92 at number of generations of 25.

GAIM16 had a similar trend. However, the average cost it obtained was also lower than that of GAARM for all generations. For example, GAIM16 obtained a cost of $415.07 at number of generation of 25, compared with $462.92 for

GAARM. The results suggest that, among the GA-based algorithms, GAIM16 had the best search ability for the air conflict avoidance scenario.

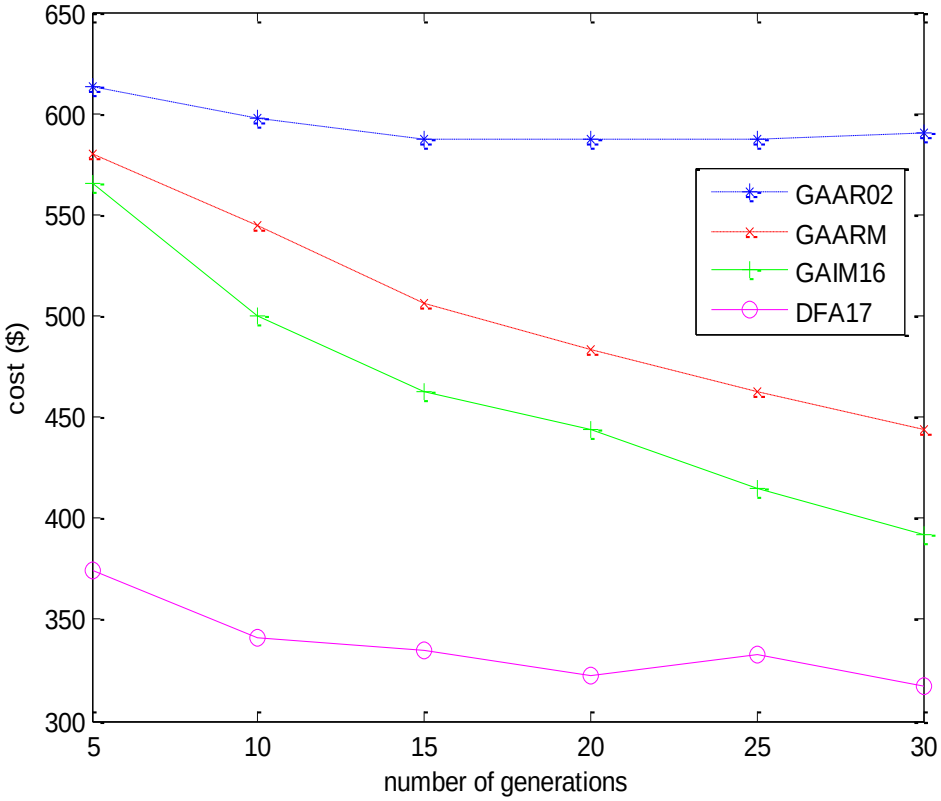


Fig. 6.19: Variation of cost for scenario with aircraft avoidance

DFA17 had an average cost of $373.97 for a generation of 5 (Fig. 6.19). This represents an improvement of $239.18, compared with the corresponding value for GAAR02. This performance by DFA17 was displayed for all the other generations. Overall, the costs obtained by DFA17 for the scenario were much lower than those of the other algorithms.

It could be observed in the Fig. 6.19 that the average costs of obtained solutions for all the algorithms decreased as the number of generations was increased. After number of generations of 15, costs produced by GAAR02 did not improve substantially. In addition, the solutions produced by GAAR02 had the worst costs for all the considered generations, compared with the other algorithms. This indicates that the algorithm was trapped at local minima. The solutions generated by GAARM had better costs and the performance of GAARM steadily improved as the number of generations was increased. A similar trend was observed for

GAIM16. However, GAIM16 produced results that were better than those of GAARM for every number of generations. DFA17 outperformed GAAR02, GAARM and GAIM16 by producing costs that were much lower. It could be observed that after number of generations of 20, DFA17 did not improve its performance much. This is most likely because the solutions it produced for previous numbers of generations were already close to the global minimum. In addition, a slight increase was observed for number of generations of 25. This is likely because of the random elements included in the initial population generation and repair functions of DFA17. It is expected that the plot would smoothen out as the number of algorithm runs tends to infinity.

Table 6.19: Cost results for aircraft avoidance scenario (number of generations of 5)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 613.15 | 33.29 | 467.86 | 633.82 |
| **GAARM** | 580.12 | 62.32 | 367.69 | 633.82 |
| **GAIM16** | 565.85 | 61.78 | 367.69 | 633.82 |
| **DFA17** | 373.97 | 88.40 | 167.36 | 633.82 |

Further results can be seen in Table 6.19. For number of generations of 5, the cost of the best solution derived by GAAR02 was $467.86. The costs of paths suggested by GAAR02 also had a standard deviation of $33.29. Both GAARM and GAIM16 had the same best cost of $367.69. However, GAIM16 had a lower standard deviation of $61.78, compared with $62.32 for GAARM. The best performance, in terms of best cost obtained, was by DFA17. This cost was $167.36. However, DFA17 had the highest standard deviation of $88.40. This is explained by the increase in the search ability of DFA17, which may temporarily worsen a solution while leaving a local optimal. The cost of worst paths produced by all algorithms was the same at $633.82.

Table 6.20: Cost results for aircraft avoidance scenario (number of generations of 15)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 587.31 | 68.90 | 267.52 | 633.82 |
| **GAARM** | 506.29 | 85.25 | 208.85 | 633.82 |
| **GAIM16** | 462.35 | 91.87 | 208.85 | 633.82 |
| **DFA17** | 334.60 | 88.88 | 167.36 | 633.82 |

At number of generation of 15 (Table 6.20), the average and best costs obtained by GAAR02 had improved to $587.31 and $267.52. Similarly, GAARM and GAIM16 had an average cost of $506.29 and $462.35 respectively. The cost of the best solution for both algorithms was $208.85. DFA17 had the overall best cost of $167.36, compared to those of other algorithms. Its performance in terms of average cost of $334.60 was also the best.

Table 6.21: Cost results for aircraft avoidance scenario (number of generations of 30)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 590.33 | 58.23 | 367.69 | 633.82 |
| **GAARM** | 444.18 | 75.89 | 167.36 | 592.33 |
| **GAIM16** | 391.96 | 82.74 | 208.85 | 626.20 |
| **DFA17** | 317.37 | 76.48 | 108.68 | 550.84 |

Comparable results were exhibited by the algorithms for number of generations of 30 (Table 6.21). With average, best and worst cost of $590.33, $367.69 and $633.82 respectively, GAAR02 had the worst performance. GAARM had obtained a best cost of $167.36, which was lower than that of GAIM16. However, GAIM16 had a better average cost of $391.96, compared with $444.18 for GAARM. The average cost of $317.37 by DFA17 was the best, compared with the other three algorithms.

The best paths obtained by the algorithms are shown in Fig. 6.20 – 6.23. All the paths avoided the weather regions. The path suggested by GAAR02 was 6-17-28-29-40-50-60-70-79-78-87-96 (Fig. 6.20). All parts of the path lies to the right of the central weather region. The path pre-emptively started the diversion process right from the start waypoint. However, it gave the adverse weather regions a margin that was very large. This led to a longer route, with attendant delay and fuel costs.

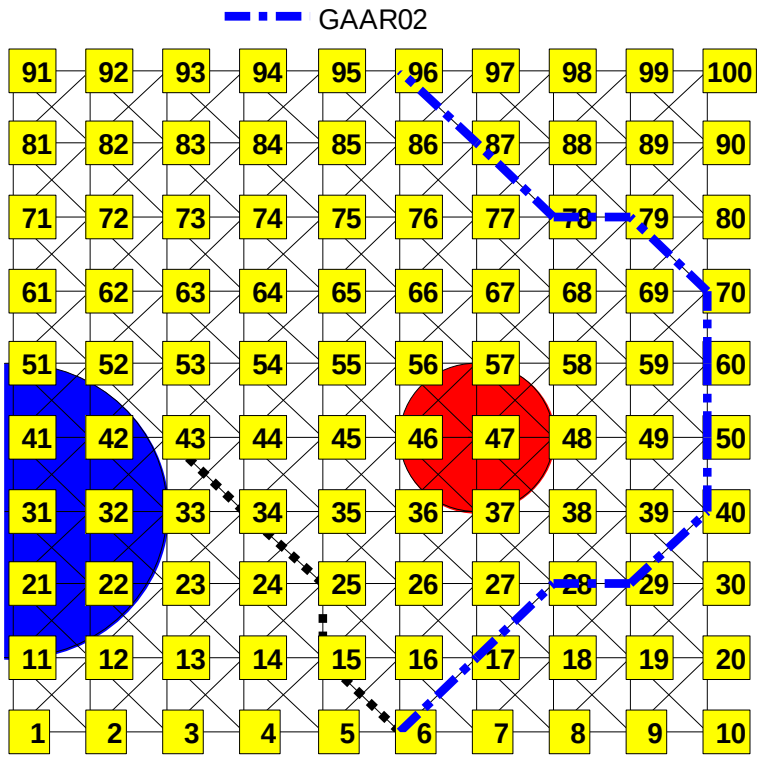


Fig. 6.20: Alternative path for aircraft avoidance scenario (GAAR02)

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted line is path of other aircraft)

The best alternative path obtained by GAARM was 6-16-26-36-45-56-66-76-85-96, as shown in Fig. 6.21. This path kept to the default path until it reached waypoint 36. It then avoided the adverse weather region. The path re-joined the default path at 56, shortly after the adverse weather region. It kept to the default path before diverting slightly to 85 and then ended at the destination. However, the generated path was much longer than those produced by the other algorithms. This indicated that GAAR02 had poor searching ability and was unable to escape from local optima.

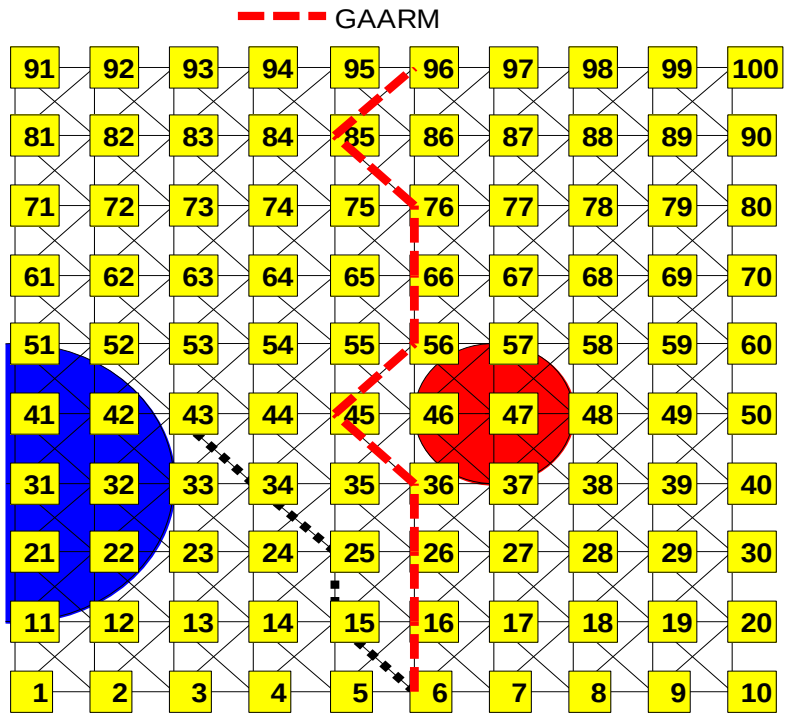


Fig. 6.21: Alternative path for aircraft avoidance scenario (GAARM)

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted line is path of other aircraft)

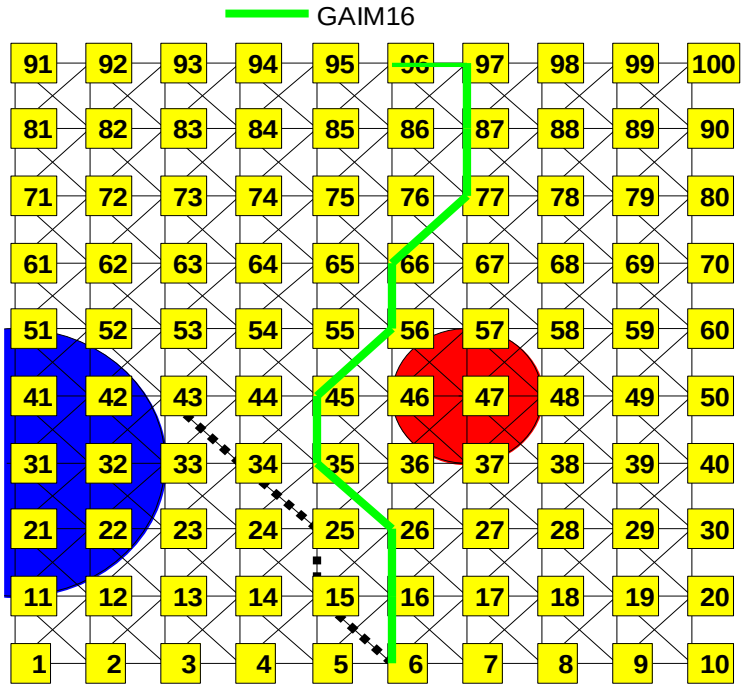


Fig. 6.22: Alternative path for aircraft avoidance scenario (GAIM16)

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted line is path of other aircraft)

The best alternative path produced by GAIM16 is shown in Fig. 6.22. The alternative path kept to the default one until it reached waypoint 26. It then diverted around the adverse weather. The path re-joined the default path at 56. However, it deviated from the default path from waypoint 66 to the destination waypoint.

Fig. 6.23 shows the alternative path generated by DFA17. The path kept to the default path until it reached waypoint 26. It then carried out a diversion around the adverse weather region. It did not return to the default path until waypoint 86. This path was the shortest among all the alternative paths produced by the four algorithms.

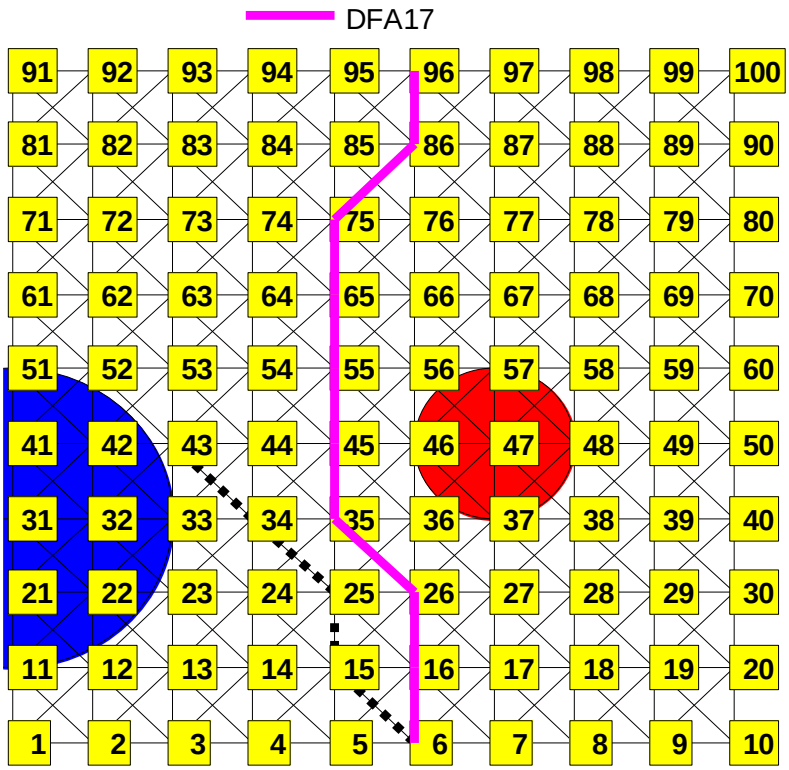


Fig. 6.23: Alternative path for aircraft avoidance scenario (DFA17)

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted line is path of other aircraft)

Compared with the scenario without another aircraft in the considered airspace, the average costs of path solutions generated in this scenario were higher. The average cost in the scenario without another aircraft was $397.46, $298.23 and $261.61, for number of generations of 30 (Table 6.22). In this scenario involving one other aircraft, the equivalent cost greatly increased by $192.87 to $590.33 for GAAR02, as shown in Table 6.21. The increases of $145.95 and $130.35 in

average costs were less for GAARM and GAIM16 respectively. On the other hand, DFA17 incurred only an increase of $49.5. It was, therefore, able to respond more robustly to the changed scenario. The increase in average costs of generated paths in this scenario is likely due to the increased complexity introduced by the existence of the other aircraft.

### 6.4.3 Scenario involving multiple aircraft

In this scenario, more than two aircraft co-existed in the airspace. The aircraft under consideration had a default path from 6 to 96. The trajectory of other aircraft were 6-15-25-34-43, 10-20-30-40-50-60-70-80-90-100, 6-16-26-36-45-54-64-74-84-94 and 6-5-4-3-12. Fig. 6.24 shows the airspace area under consideration. The first waypoint indicates the start waypoint at beginning of simulation time. The aircraft had average speeds of 1000 KM/hr. The simulation parameters are the same as those of the previous scenario (Section 6.4.2).

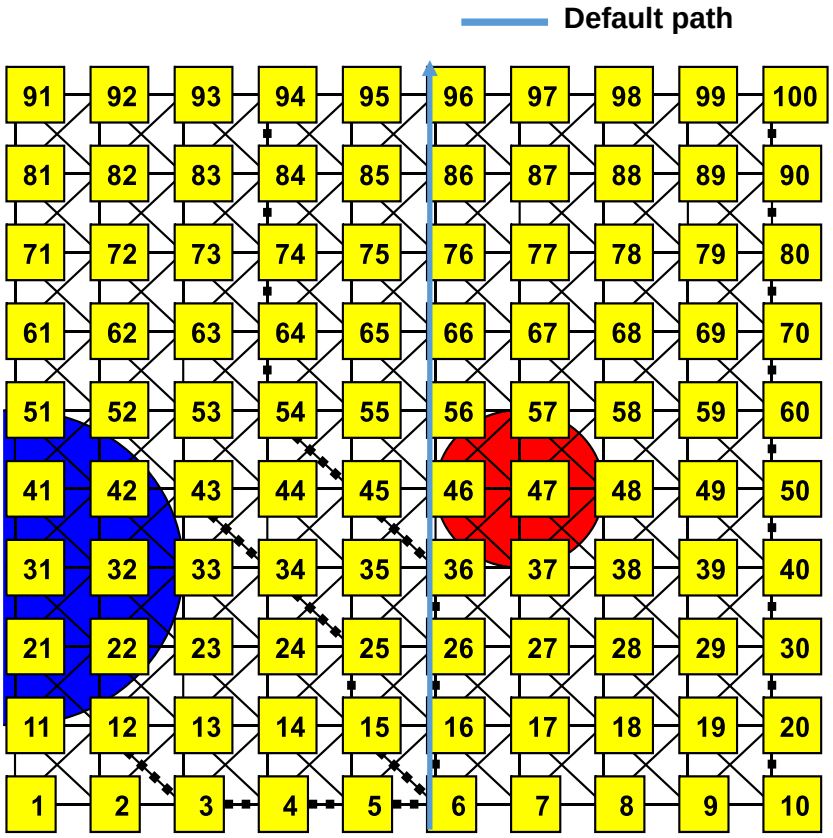


Fig. 6.24: Airspace scenario with multiple aircraft

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted lines are paths of other aircraft)

Fig 6.25 shows the convergence of costs of paths obtained by the considered algorithms. From a value of $1326.79 for GAAR02 at number of generations of 5, the average cost improved to $933.14 at number of generations of 30. Similarly, GAARM was from $1092.24 and $555.65 at number of generations of 5 and 30 respectively. This indicates that GAARM had a better ability to search for suitable solutions in this more complex scenario. However, the performance of GAIM16 was better than both algorithms. Its corresponding average costs were $973.27 and $449.58 for number of generations of 5 and 30 respectively.

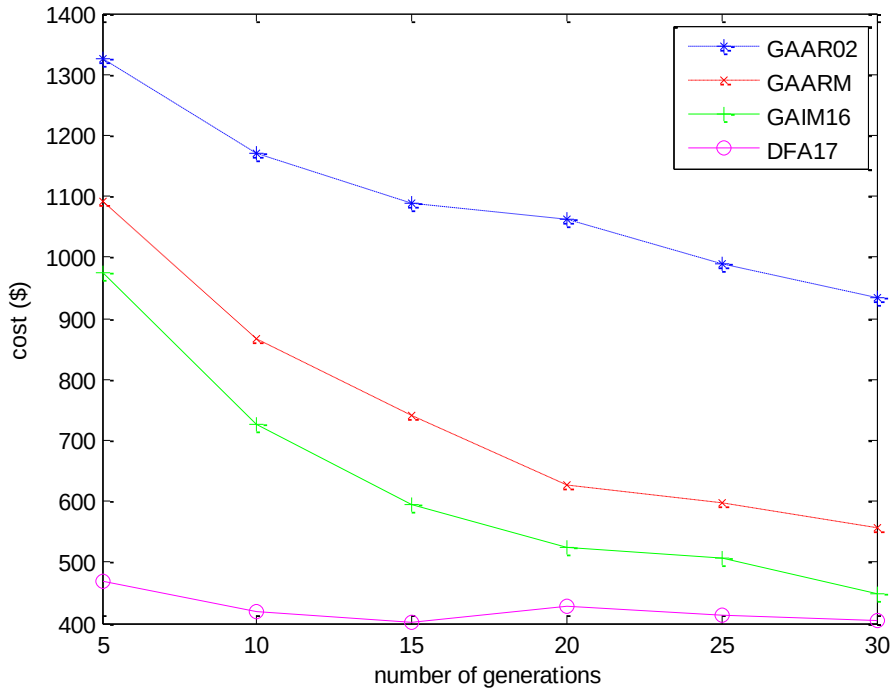


Fig. 6.25: Variation of costs for scenario with multiple aircraft

As shown in Fig. 6.25, there were little changes in costs for DFA17 for number of generations from 10. At number of generations of 20 and 25, there were slight increases in solution generated. This is likely due to the random nature of the initial firefly generation and repair functions of algorithm. The overall trend for GAAR02, GAARM and GAIM16 showed steady decrease (improvements) in costs with increase in number of generations. From the figure, it can be observed that the performance trends for GAARM and GAIM16 were the closest, in terms of additional decrease in average cost due to each increment of generation number. However, for all the number of generations, GAIM16 had lower costs compared with GAARM. This implied it had a superior search ability compared

with GAARM. The largest improvements for GAAR02 were between number of generations of 5 and 15. In addition, it had solutions that were much higher than those of GAARM and GAIM16 for all the number of generations. These results indicate that GAAR02 had inferior search ability compared with GAARM and GAIM16.

Table 6.23 shows further details for the simulation scenario. The cost of the best path produced by GAARM and GAIM16 was both $367.69 at number of generations of 5. The worst performance was by GAAR02 at $568.03. DFA17 performed best and had best solution cost of $226.03. The standard deviation for GAAR02, GAARM, GAIM16 and DFA17 were $273.46, $295.08, $273.31 and $93.86 respectively. The cost of the worst (longest) solution was $1776.32 for GAAR02, GAARM and GAIM16. DFA17 had a better performance of $726.87 in this aspect.

Table 6.23: Cost results for multiple aircraft scenario (number of generations of 5)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 1326.79 | 273.46 | 568.03 | 1776.32 |
| **GAARM** | 1092.24 | 295.08 | 367.69 | 1776.32 |
| **GAIM16** | 973.27 | 273.31 | 367.69 | 1726.32 |
| **DFA17** | 468.42 | 93.86 | 226.03 | 726.87 |

Table 6.24 shows the performance of the algorithms for number of generations of 30. All the algorithms improved in terms of average costs, standard deviation, best costs and worst costs. The respective average costs for GAAR02, GAARM, GAIM16 and DFA17 were $933.14, $555.65, $449.58 and $403.60. The relative performance by the algorithms were maintained. GAIM16 remained the second-best and DFA17 had the best performance, in terms of average cost. This trend was also observed for the standard deviation of costs for the algorithms. The standard deviation was $297.60, $184.29, $123.72 and $81.35 for GAAR02, GAARM, GAIM16 and DFA17 respectively. DFA17 had the least change in average costs. However, it had the lowest value at both number of generations and was already the closest to the optimal solution.

Table 6.24: Cost results for multiple aircraft scenario (number of generations of 30)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 933.14 | 297.60 | 284.71 | 1684.83 |
| **GAARM** | 555.65 | 184.29 | 226.03 | 1242.14 |
| **GAIM16** | 449.58 | 123.72 | 208.85 | 892.16 |
| **DFA17** | 403.60 | 81.35 | 208.85 | 626.20 |

From Table 6.24, the best costs obtained by GAAR02 and GAARM were $284.71 and $226.03. GAIM16 and DFA17 had the same best cost of $208.85. The contributors to the cost of the best alternative path by GAAR02 were flight delay, missed connections and additional fuel burn costs. The component costs were $39.76, $50.00 and $194.95 respectively. For GAARM, the corresponding costs for the flight delay, missed connections and additional fuel burn components were $29.82, $50.00 and $146.21. The contributors to the best costs for GAIM16 and DFA17 were flight delay, missed connections and additional fuel burn components at $26.91, $50.00 and $131.93 respectively.

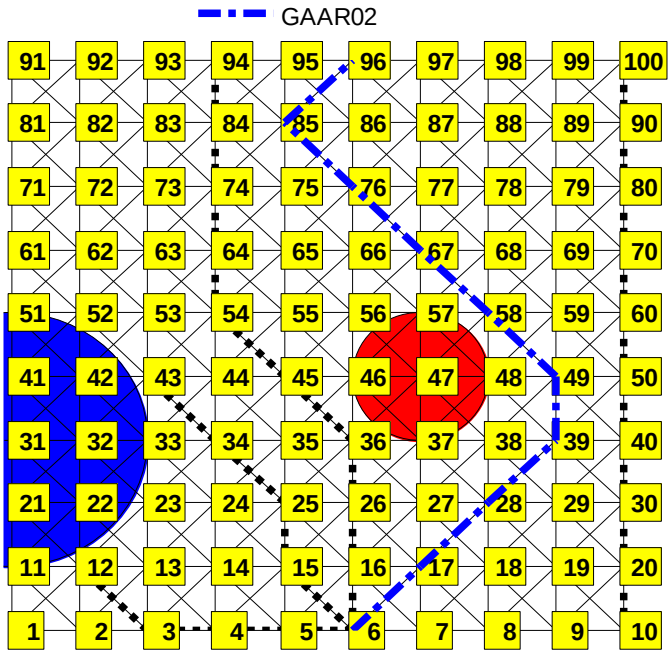


Fig. 6.26: Alternative path for multiple aircraft scenario (GAAR02)

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted lines are paths of other aircraft)

Fig. 6.26 shows the best alternative path generated by GAAR02. The path was represented as 6-17-28-39-49-58-67-76-85-96. The path was to the right side of the smaller weather region. It can be observed that the route avoided the airspace section with higher number of aircraft. However, this produced a route with higher costs. In addition, the path did not keep to the default path for most parts. It briefly crossed the default path at waypoint 76. The path avoided all the adverse weather regions.

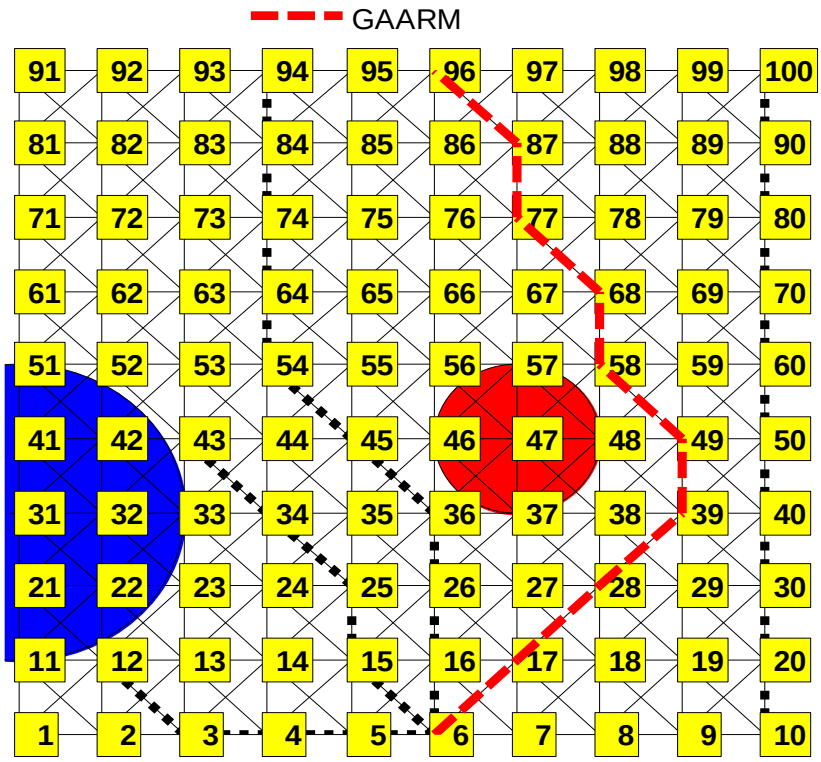


Fig. 6.27: Alternative path for multiple aircraft scenario (GAARM)

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted lines are paths of other aircraft)

Similarly, the path produced by GAARM avoided all the weather regions (Fig. 6.27). It kept to the right of the smaller weather region. The left hand side of the airspace with a higher number of aircraft was avoided. However, this involved an increased route costs. In addition, the path did not use any section of the default path. This has the potential of greater disruption to usual air traffic.

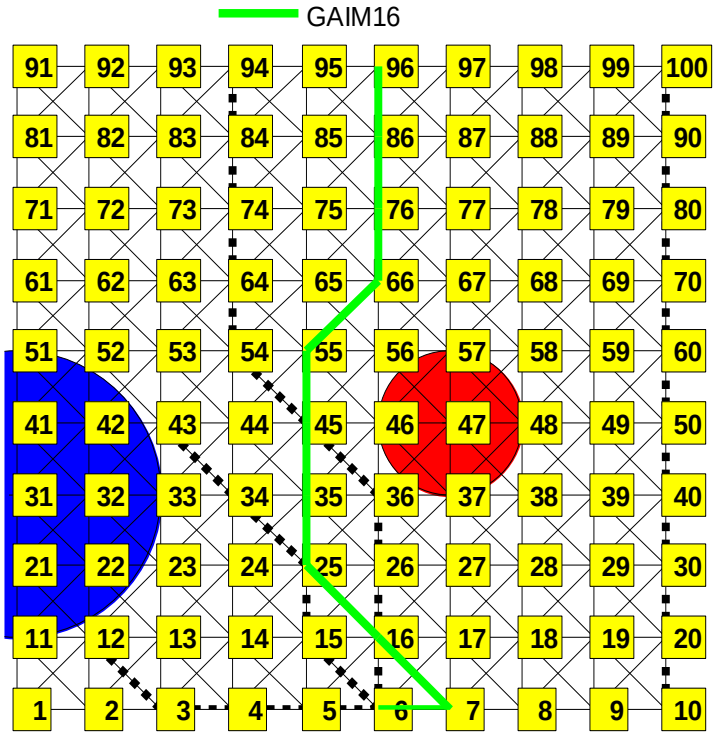


Fig. 6.28: Alternative path for multiple aircraft scenario (GAIM16)

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted lines are paths of other aircraft)

The alternative path derived by GAIM16 was 6-7-16-25-35-45-55-66-76-86-96, as shown in Fig. 6.28. Compared with GAARM, it was located in the area between the two weather regions. The area also had higher number of aircraft, compared with the right side of the airspace. However, the generated path had a lower cost of $208.85, compared with $226.03 for GAARM. The alternative path initially deviated from the default path to avoid other aircraft. The adverse weather regions were also avoided. The alternative path then re-joined the default path form waypoint 66.

Similar approach was also seen in the path obtained by DFA17 (Fig. 6.29). The path had cost equal to that obtained by GAIM16. There were some minor differences in the two paths before waypoint 66. In order to avoid other aircraft, the alternative suggested by DFA17 first deviated to waypoint 17. It then returned to the default path at waypoint 16. It again deviated from the default path to avoid the smaller weather region. It returned to the default path from waypoint 66.

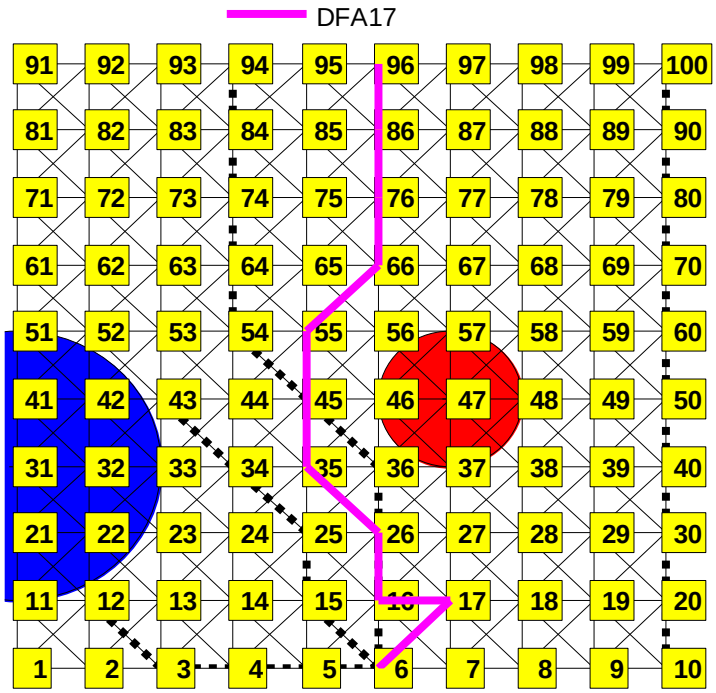


Fig. 6.29: Alternative path for multiple aircraft scenario (DFA17)

(where yellow rectangles represent waypoints, black lines are route links, red and blue circular regions are weather regions of high and medium intensities respectively, black dotted lines are paths of other aircraft)

Overall, it was observed that the algorithms were able to generate alternative paths that avoid other aircraft in the considered airspace. Regions of adverse weather were also avoided. The contributors to the costs of alternative paths were flight delay, missed connections and fuel burn costs. The best performance was obtained by GAIM16 and DFA17 with lower costs.

### 6.4.4 Scenario involving flight level changes

In this scenario, the aircraft could make changes to flight level during the weather avoidance process. The difference between admissible flight levels in one direction of flight was approximately 2000 feet (CAA 2015). A three-dimensional grid model of the airspace was generated and used (Gleich 2008). To indicate possible flight links, adjacent and diagonal waypoints were interconnected (Fig. 6.30). Each waypoint on the flight levels was a vertical projection of the waypoint located directly above it. To reduce computation, the model considered flight movements within five level offsets above a base flight level of FL250.

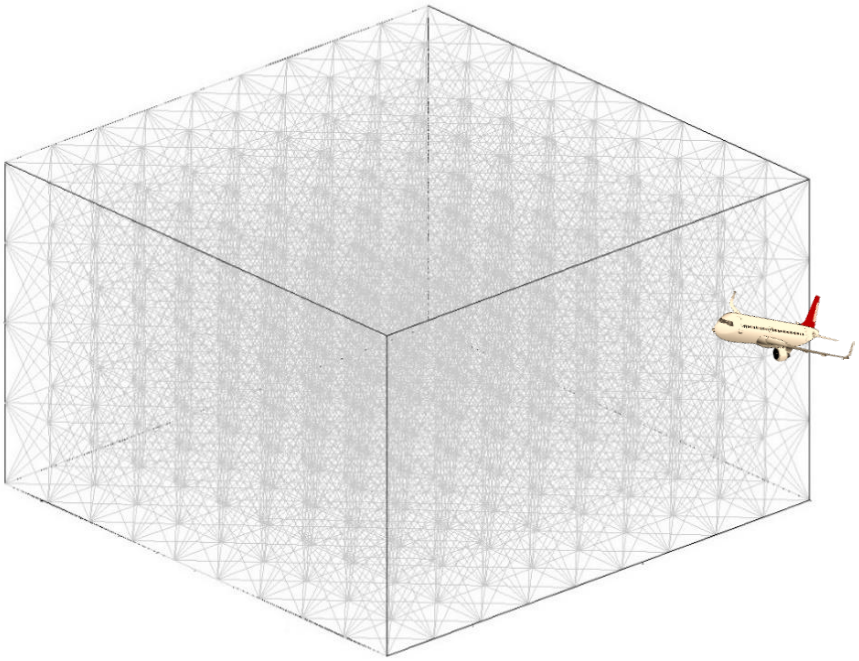

Fig. 6.30: Three-dimensional airspace model

Two weather regions of medium and high intensities were considered. The adverse weather regions were located on waypoint 31 and 47 respectively. The locations are analogous to the two-dimensional case discussed (Section 6.9.3), except that altitude was considered in this scenario. Both weather regions were located on FL290. The radii of the regions were 100 and 50KM respectively. The aircraft trajectories were also the same as the two-dimensional scenario. Each component cost of flight delay, weather impact, missed connections and flight level changes had a weight of 1. Additional fuel burn cost had a weight of 57 to penalise high fuel use. The default path of the considered aircraft was 6-16-26-36-46-56-66-76-86-96, a straight route between waypoints 6 to 96 on FL290. The simulation parameters are shown in Table 6.8.

Fig. 6.31 shows the variation of average costs for the scenario with support for flight level changes. GAIM16 had an average cost of $1436.01 and $845.34 for number of generations of 10 and 50 respectively. This was much better than GAARM, which had an average cost of $1675.06 and $1097.25 for number of generations of 10 and 50 respectively.

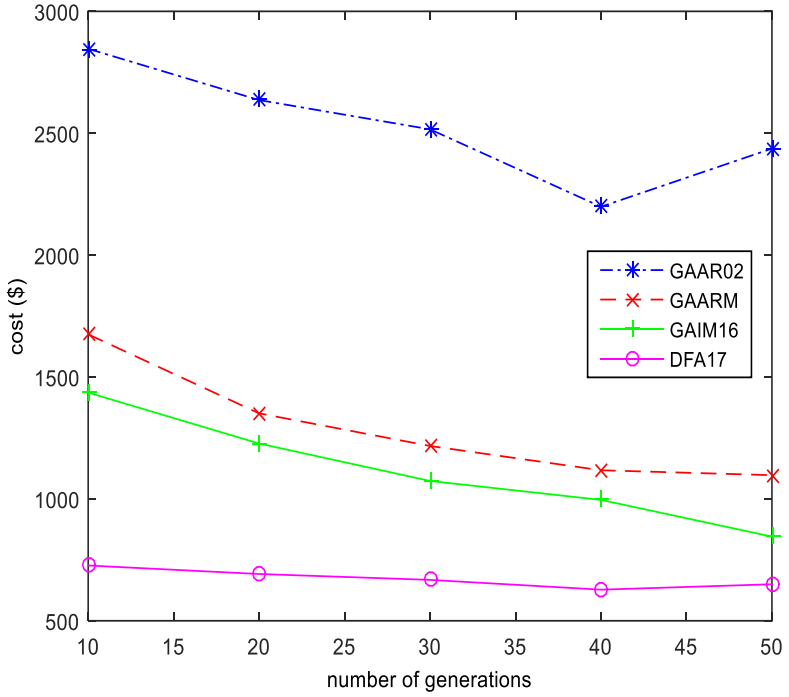


Fig. 6.31: Variation of costs for scenario with flight level changes

The overall trend for the four algorithms was that the decrease of the costs of obtained routes was associated with increased numbers of generations. DFA17 obtained the lowest costs, compared with the rest of the algorithms. It had the least change in costs as the number of generations was increased. Overall, GAAR02 had the highest changes in costs as number of generations was increased from 10 to 40. However, it had the worst performance for all the number of generations. In spite of the overall decreasing trend, there was an increase in the obtained average costs at number of generations of 50 for GAAR02. This is most likely due to the probabilistic components of the algorithm such as mutation and crossover functions. It is expected that the plot would smoothen out (decreasing throughout) as number of runs of the algorithm is increased. Both GAARM and GAIM16 showed relatively steady decrease in costs as number of generations are increased.

Further simulation results are shown in Table 6.25 for number of generations of 10. GAARM produced paths with average costs of $1675.06 and standard deviation of $647.41. The best performance in this regard was the average cost of $726.75 by DFA17. The standard deviation of the solution costs by DFA17 was $137.61. GAAR02 had an average cost of $2846.38 and standard deviation of $1466.01. This was almost four times that of DFA17 and represents the worst

performance. GAIM16 had the second-best performance in terms of average cost of $1436.01.

Table 6.25: Cost results for scenario with flight level changes (number of generations of 10)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 2846.38 | 1466.01 | 950.67 | 5628.97 |
| **GAARM** | 1675.06 | 647.41 | 709.18 | 3969.15 |
| **GAIM16** | 1436.01 | 444.10 | 609.01 | 2826.48 |
| **DFA17** | 726.75 | 137.61 | 326.03 | 1100.34 |

The best cost of $326.03 was produced by DFA17, as shown in Table 6.25. The components of the cost are $26.91, $50.00, $100.00 and $131.93 costs for flight delay, missed connections, level changes and additional fuel burn respectively. The corresponding component costs for GAARM for its best solution were $60.85, $50.00, $300.00 and $298.33, giving a total of $709.18. For GAIM16, the corresponding values were $43.88, $50.00, $300.00 and $215.13 for a total cost of $609.01. With costs of $67.88, $50.00, $500.00 and $332.79 for flight delay, missed connections, level changes and additional fuel, GAAR02 had the highest flight level change costs.

Table 6.26: Cost results for scenario with flight level changes (number of generations of 30)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 2516.04 | 1532.98 | 709.01 | 5628.97 |
| **GAARM** | 1217.02 | 354.55 | 308.85 | 2443.50 |
| **GAIM16** | 1073.59 | 281.57 | 467.36 | 1934.66 |
| **DFA17** | 667.81 | 151.88 | 308.85 | 1250.67 |

Similar observations were made on performance results for number of generations of 30 (Table 6.26). With the increase in number of generations, average and best costs dropped for all the algorithms. GAARM had a lower best cost of $308.85 with a larger standard deviation of $354.55, compared with GAIM16. However, the top two performance in terms of average costs were by GAIM16 and DFA17. The algorithms produced paths with average costs of $1073.59 and $667.81 respectively. The standard deviations were $281.57 and $151.88 for GAIM16 and DFA17 respectively.

Table 6.27: Cost results for scenario with flight level changes (number of generations of 50)

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 2437.05 | 1402.22 | 850.67 | 5628.97 |
| **GAARM** | 1097.25 | 337.64 | 326.03 | 2093.67 |
| **GAIM16** | 845.34 | 257.06 | 367.36 | 1551.68 |
| **DFA17** | 649.84 | 157.86 | 308.85 | 1209.35 |

For number of generations of 50, further results are shown in Table 6.27. Average costs had improved to $2437.05, $1097.25, $845.34 and $649.84 for GAAR02, GAARM, GAIM16 and DFA17 respectively. This indicated that, on the average, the costs of paths generated by GAAR02 were nearly four times of those obtained by DFA17. The corresponding standard deviations were $1402.22, $337.64, $257.06 and $157.86 for GAAR02, GAARM, GAIM16 and DFA17 respectively. DFA17 had the best performance in terms of average cost for the number of generations. This indicates that it was able to cope with the increased search complexity that includes the flight level change options, avoidance of other aircraft and avoidance of multiple adverse weather regions. The costs of the best paths obtained by GAAR02, GAARM, GAIM16 and DFA17 were $850.67, $326.03, $367.36 and $308.85 respectively. These best paths are shown in Fig. 6.32 – 35.

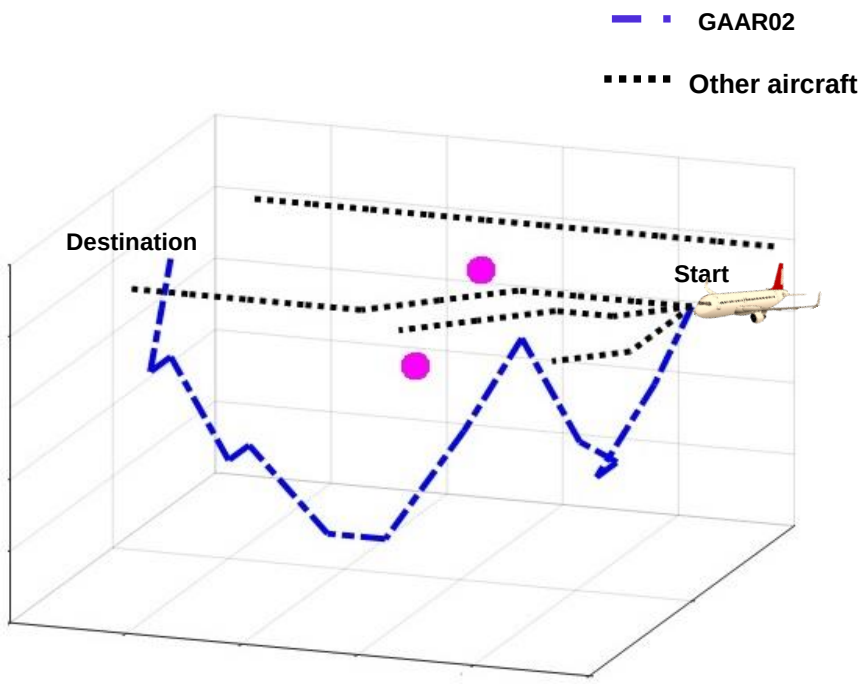


Fig. 6.32: Alternative path for flight level change scenario (GAAR02)

All the paths generated by the algorithms were able to avoid the regions of adverse weather and other aircraft. Fig. 6.32 shows the best path generated by GAAR02 for generation number of 50. The centres of the regions of adverse weather are shown as the magenta dots in the figure. The path generated was below the default flight level. One disadvantage of the suggested alternative path was the large number (eight) of flight level changes.

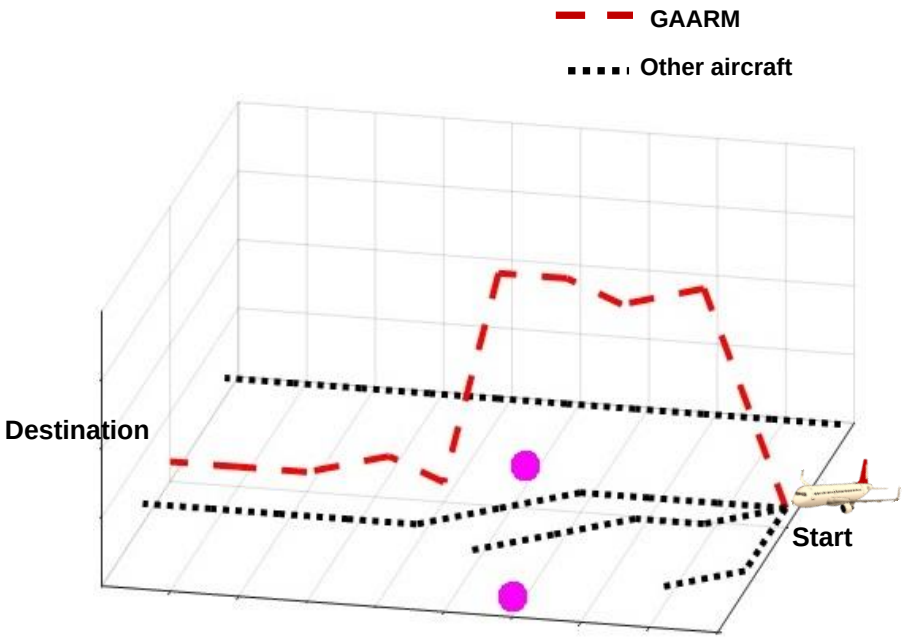


Fig. 6.33: Alternative path for flight level change scenario (GAARM)

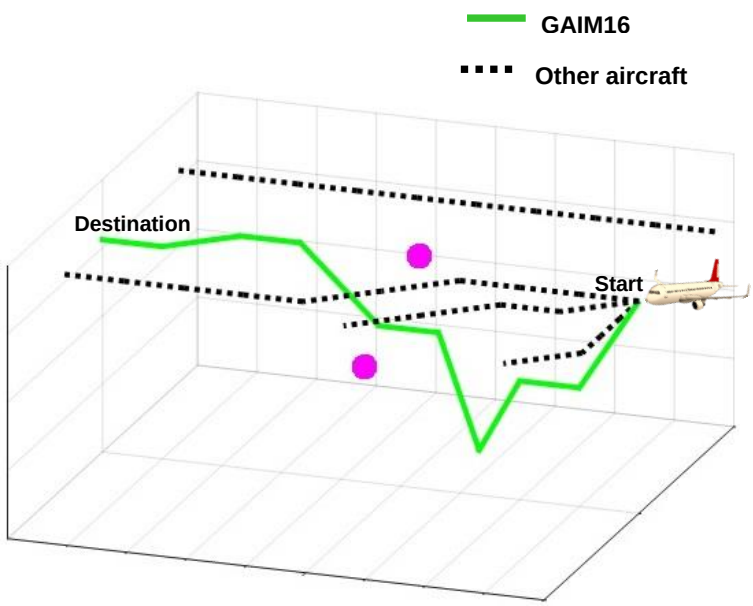


Fig. 6.34: Alternative path for flight level change scenario (GAIM16)

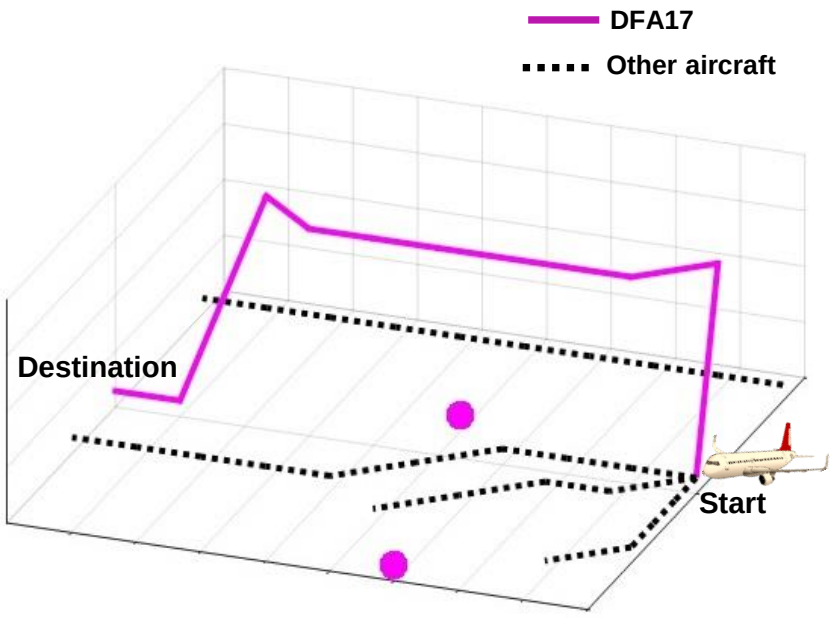


Fig. 6.35: Alternative path for flight level change scenario (DFA17)

On the other hand, the path suggested by GAARM had only two level changes. The alternative route involved changing to flight levels above the regions with adverse weather and large numbers of other aircraft. GAIM16 avoided the adverse weather regions and other aircraft by going to flight levels below them.

The path involved four flight level changes, a value that is higher than that of GAARM. The alternative path generated by DFA17 involved two flight level changes. The change to a higher flight level allowed the path to avoid the adverse weather regions and other aircraft. It returned to the default level shortly before the destination waypoint.

## 6.5 Processing times of the algorithms

This section compares the proposed algorithms by measuring their processing times for the considered scenarios. The average values for each algorithm run are presented in Table 6.28.

Table 6.28: Processing times for considered scenarios

| Scenario | Processing time for each run (seconds) | | | |
|---|---|---|---|---|
| | **GAAR02** | **GAARM** | **GAIM16** | **DFA17** |
| Section 6.3.2: Effects of mutation rate (Benchmark 1) | 4.3933 | 10.8764 | 14.2002 | -- |
| Section 6.3.3: Variation of population size for Benchmark 1 | 0.6151 | 1.1272 | 1.3844 | 23.4957 |
| Section 6.3.4: Effect of number of generations (Benchmark 1) | 2.3476 | 4.2472 | 5.2317 | 9.7304 |
| Section 6.3.5: Effect of an adverse weather region (Benchmark 1) | 1.1532 | 1.6700 | 1.9457 | 6.6529 |
| Section 6.3.6: Effect of multiple adverse weather regions for Benchmark 1 | 1.1260 | 1.6443 | 1.9196 | 6.3812 |
| Section 6.4.1 Effects of multiple weather regions for grid-based scenario | 1.6728 | 2.5605 | 2.9816 | 21.0432 |
| Section 6.4.2 Scenario considering aircraft avoidance | 1.7607 | 2.6484 | 3.0547 | 22.4665 |
| Section 6.4.3 Scenario involving multiple aircraft | 2.3874 | 3.0954 | 3.4727 | 17.5105 |
| Section 6.4.4 Scenario involving flight level changes | 9.0720 | 13.7120 | 21.0980 | 779.4620 |

The stated times are in terms of central processing unit (CPU) clock time. It could be seen that for the scenario in Section 6.3.2, which considered effects of mutation rate (Benchmark 1) for the algorithms, the processing times for GAAR02, GAARM and GAIM16 were 4.3933, 10.8764 and 14.2002 seconds (s) respectively. From these values, it can be observed that GAIM16 took about 6s more than GAAR02 whereas GAIM16 took almost 10s more than GAAR02. The increase in processing times for GAAR02 and GAIM16 was likely due to the modifications to the mutation operator and the time it took to repair reroutes.

A similar trend was observed for the processing times for the scenario in Section 6.3.3. Specifically, the algorithms GAAR02, GAARM, GAIM16 and DFA17 had processing times of 0.6151s, 1.1272s, 1.3844s and 23.4957s respectively. It could be seen that GAARM had 0.5121s processing time than GAAR02. However, as earlier discussed GAARM produced solutions with lower average costs compared with GAAR02. GAIM16 had only 0.2572s more processing time compared with GAARM, despite performing better than GAARM. The highest processing time was observed for DFA17, although it had the best performance in terms of average costs. This indicates that its superior performance was associated with increased processing. Its larger processing time was likely due to the increased search by the proposed movement operator.

Similar trends could be observed for the scenario of Section 6.3.4 (effect of number of generations for Benchmark 1). It could also be observed that the increase in number of adverse weather regions from one (Section 6.3.5) to three regions (Section 6.3.6) did not substantially affect the processing time for each of the algorithms. For example, the processing times for GAIM16 in the two scenarios were 1.9457s and 1.9196s respectively. The slight reduction in time could be due to the random elements of the algorithms, such as the initial population generation function.

For the case that considered the effects of multiple weather regions for grid-based scenario (Section 6.4.1), the processing times were 1.6728s, 2.5605s, 2.9816s and 21.0432s for GAAR02, GAARM, GAIM16 and DFA17 respectively. It could be seen that all algorithms had increased processing times, compared with the Benchmark 1 scenario (Section 6.3.6). This is expected as the scenario in Section 6.4.1 had a larger number of waypoints (100).

The processing times for the scenario with the avoidance of an aircraft (Section 6.4.2) were 1.7607s, 2.6484s, 3.0547s and 22.4665s for GAAR02, GAARM, GAIM16 and DFA17 respectively. This represents an increase of 0.0879s, 0.0879s, 0.0731s and 1.4233s for GAAR02, GAARM, GAIM16 and DFA17. It can be observed that the processing time of GAIM16 was the least affected. This shows that the algorithm is able to effectively handle the increased complexity of the airspace.

When the number of aircraft in the airspace is increased (Section 6.4.3), processing time increased to 2.3874s, 3.0954s and 3.4727s for GAAR02, GAARM and GAIM16 respectively. This corresponds to an increase of 0.6267s, 0.447s, and 0.418s, compared with the single aircraft avoidance scenario. Among the three GA-based techniques, GAIM16 was the least affected by the increased number of aircraft in the considered airspace. This indicates that it had improved ability to search the complex solution space. DFA17 had slightly lower processing time of 17.5105s, compared with the single aircraft avoidance scenario. This decrease could be accounted for by the probabilistic nature of the repair and initial population generation functions of DFA17. To address this, the algorithm could be run more times when the average processing time is calculated. However, this would likely involve very large simulation time.

For the scenario with flight level changes (Section 6.4.4), the processing times were 9.0720s, 13.7120s, and 779.4620s respectively for GAAR02, GAARM and DFA17. It could be seen that the least processing time was obtained by GAAR02. However, GAAR02 also produced solutions with the worst costs. GAARM had slightly higher processing costs, but produced solutions with lower costs. DFA17 had the highest processing time while producing reroutes with the best costs. GAIM16 had a much lower processing time of 21.0980s, but had the second best performance in terms of average costs. Therefore, GAIM16 produced reroutes with the best trade-off between processing time and route cost.

## 6.6 Limitations and constraints of the algorithms and platform

One limitation of the considered algorithms is the assumption of the existence of route network data about the considered airspace. However, the work can be extended to cover the situation where this information is not available. This can

be done by superimposing a virtual grid on the airspace and specifying the coordinates of the start and destination waypoints. The proposed techniques can then be applied.

Moreover, the work has used a discrete approach, in contrast to a continuous system. The discrete technique was chosen because of its better fitting to existing route graph network, compared with continuous models. In addition, with discrete graph networks, computational loads could be reduced. This because fewer data points are searched in discrete network scenarios.

Another constraint was that the average of speed on a flight link was considered. Effects of changes in instantaneous speeds within a flight link was not considered. The work assumes that the recommended range of aircraft speed is available before the algorithms are run. It is expected that future work would handle these limitations.

In addition, for added safety, the results of the algorithms and platform would need to be verified by pilots. Also, air traffic controllers would need to approve flight reroutes before the reroutes are carried out.

## 6.7 Summary

This chapter has presented the results of simulations of the flight path rerouting scenarios. Scenarios involving regions of adverse weather have been presented. The impact of air traffic on the rerouting process has also been investigated. The results showed that the proposed algorithms had better performance than GAAR02. In most cases, DFA17 had the best performance, in terms of average cost, best cost, standard deviation, rate of convergence, and probability of optimality. The second-best performance was by GAIM16. For a small benchmark problem, GAIM16 produced results that were slightly better than those of DFA17 after a relatively larger number of generations. The superior performance of the two algorithms was likely due to their improved ability to search for solutions and escape local optima. Both algorithms proposed routes that had reduced weather impact on passengers, shorter flight delays and reduced missed connections.

# Chapter 7 : CASE STUDY – SIMULATION RESULTS USING UK WAYPOINTS

## 7.1 Introduction

This chapter presents and discusses results for a case study in the UK airspace. The case study considered flight path rerouting during adverse weather. The performance of the genetic and firefly algorithm-based techniques for flight rerouting were evaluated and discussed.

## 7.2 Case study setup and data sources

As shown in Fig. 7.1, the case study uses data sources such as aeronautical data, flight data and weather data. These data sources are discussed below.

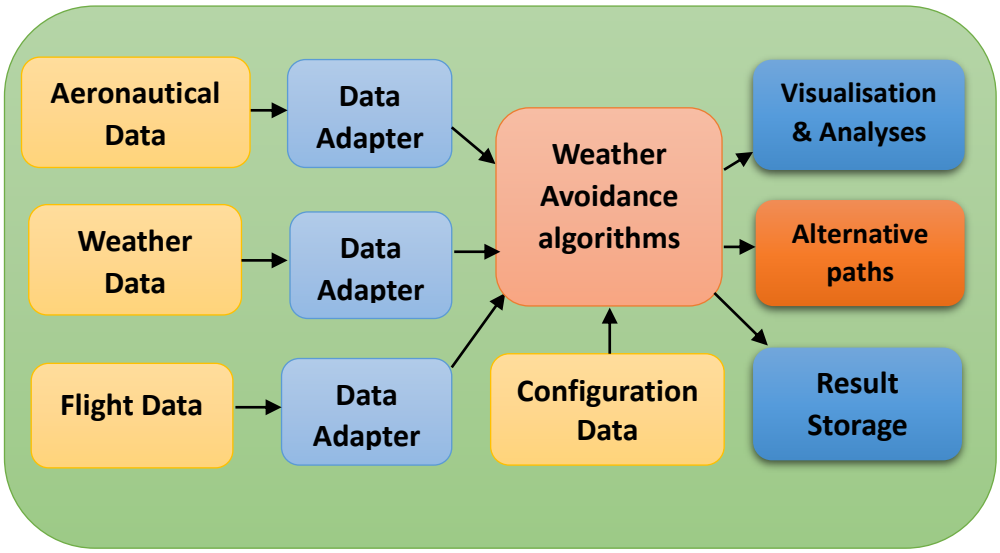


Fig. 7.1: Architecture for case study

Major aeronautical data used was waypoints data. Waypoints are key features of an airspace and include locations of navigational aids and fixes. For this scenario, waypoint information, such as geographical coordinates, were obtained from the UK Aeronautical Information Publication (AIP) (NATS 2017).

Air routes are also specified in the AIP document as series of waypoints. Upper ATS refer to routes that are located at 250 and 450 flight levels (NATS 2017). The upper ATS routes as specified in the AIP document were used to build the airspace network considered. Algorithms were written to extract this information from the document. For the weather avoidance scenario, the routes were taken

as undirected. The obtained network is shown in Fig. 7.2. The blue asterisks represent waypoints whereas the black lines represent air ways.

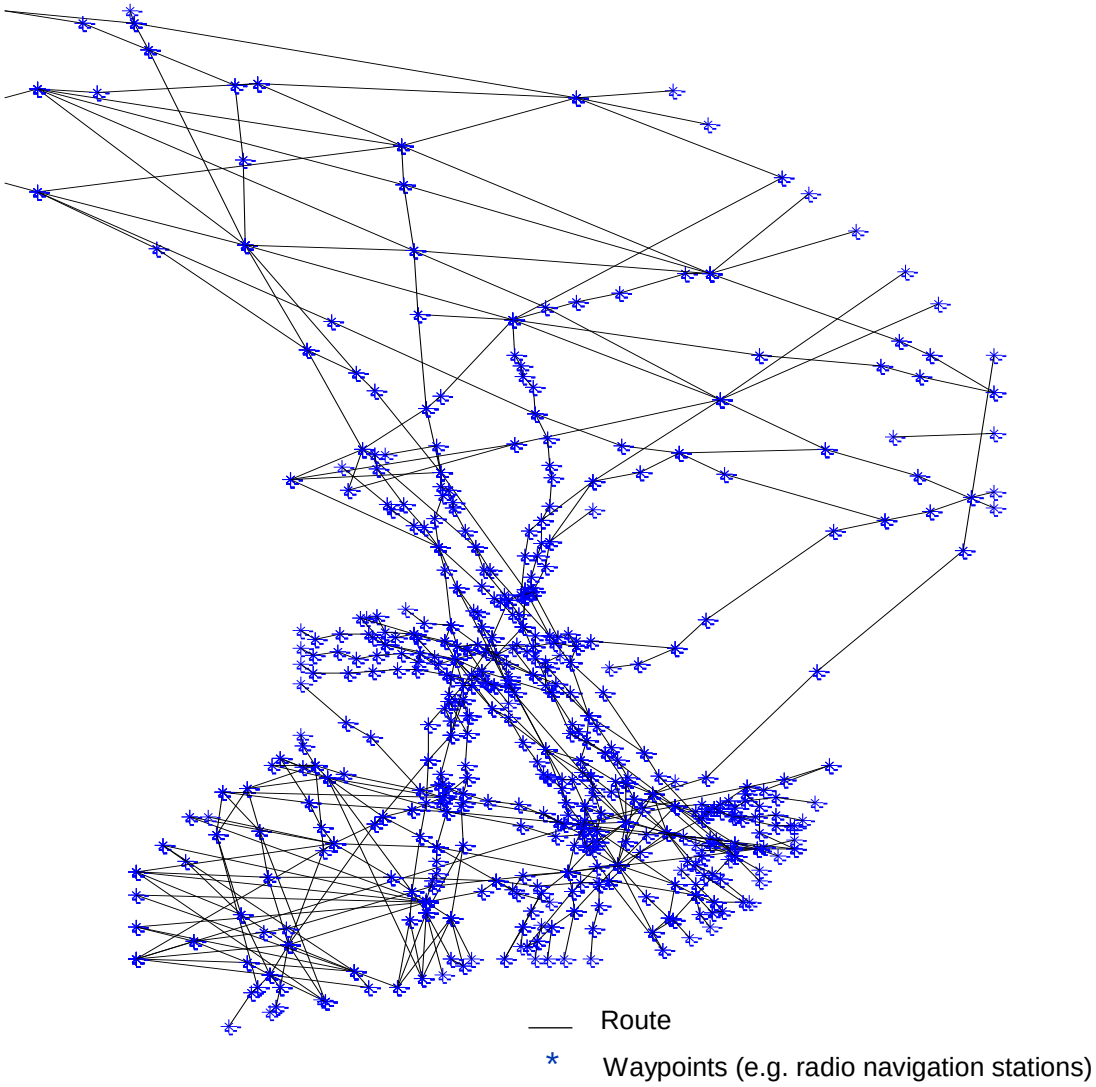


Fig. 7.2: Air route network for case study

Standard instrument departure (SID) and arrival fixes for departure and destination airports were taken from the Standard Routes Document (SRD) (NATS 2017). They serve as preferred entry points into and from the airspace network for the considered airports. In addition, some direct connections between waypoints were specified for preferred routes between the airports by the SRD. These connections were included in the input airspace network.

Data adapters acquire aeronautical data and convert them to a data types suitable for input into the rerouting algorithm. For instance, airport information, such as coordinates of airports were acquired using a RESTful interface (airport-

data.com). The airport data was in JSON format (Fig. 7.3). The JSON data was converted using the JSONlab library (Fang 2017) into a structure data type for processing in MATLAB.


Fig. 7.3: Airport information in JSON format

The flight data module provided features of the flight used for the study. For the considered flight, the average route distance, average aircraft speed, scheduled departure and arrival times were obtained from the FlightAware website (flightaware.com).

Configuration data were the inputs that specified the parameters of the rerouting algorithms, such as mutation rate, population size and crossover probability. The values for the model parameters were as defined in Chapter 4. That is, the unit missed connection cost $C_{Mc}$ was given a value of \$50 and the unit delay cost $C_{\delta}$ was given a value of \$2 (Arıkan *et al.* 2016). Similarly, the unit weather impact cost $C_m$ was \$50. The value for fuel burn per passenger-km ($b/N_p$) was \$0.0103. The specified configuration data also included the weights used for component costs. For the case study, no flight level changes were considered. Therefore, the weight for flight level costs was set to zero. The weight for each of the other component costs was 1.

Visualisation was done using the MATLAB environment and included the display of alternative paths and the airspace. The Google Map background to the figure was produced using the plot_google_map library for MATLAB (mathworks.com). Locations of weather cells were also drawn. The simulation data and results were also stored in files for further analyses.

## 7.3 Scenario with single weather region

In the scenario, a region of adverse weather centred at the waypoint with longitude of 55.0558 and latitude of -3.3458 was considered. The region had a diameter of 6 km. To obtain paths that avoid this region, the implemented algorithms were applied. Before determining waypoints impacted by the adverse weather, longitudes and latitudes of waypoints were converted to Cartesian coordinates using the Geodetic Toolbox for MATLAB (mathworks.com). The aircraft under consideration was assumed to have a start point of Edinburgh Airport (ICAO code of EGPH) and destination of London Heathrow Airport (ICAO code of EGLL). The recommended departure fixes for EGPH and EGLL in the UK SRD (NATS 2017) were adhered to.

The average aircraft speed was taken to be 457 km/hr (flightaware.com). Other parameters for the scenario are shown in Table 7.1. The scenario was run 50 times for the considered generation size of 40.

Table 7.1: Scenario parameters for case study (one weather region)

| **Parameter** | **Value** |
|---|---|
| Crossover rate | 0.9 |
| Mutation rate | 0.1 |
| Population size | 10 |
| Number of generations | 40 |
| Number of fireflies | 10 |
| $\gamma$ | 0.95 |

Fig. 7.4 shows the locations of the adverse weather region and the best alternative path produced by the discrete firefly algorithm-based technique (DFA17). The adverse weather region is shown as a circular blue region at the top left of the figure. The alternative path is shown as a solid magenta line, whereas the default path as specified in the UK SRD is shown by the dashed and dotted cyan line. The alternative path passes through waypoints TLA, IPDOR, ESKDO, INREV, UTOGU, INPIP, ABEVI, SHAPP, ERGAB, RIBEL, CROFT, BARTN, LOVEL, LISTO and HON.

The alternative path was able to avoid the region of adverse weather completely. In addition, the additional cost due to alternative path was low at $0.30 per passenger. This cost was contributed by additional fuel burn due to the increased distance of the alternative path. The reroute also allowed the aircraft to reach its destination by the scheduled arrival time. Table 7.2 shows further details on the simulation. The average cost for DFA17 over all the runs was $31.24 with a standard deviation of $33.57.


Fig. 7.4: Alternative path for scenario with singe weather region

Table 7.2: Details of results for single weather region

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 59.85 | 38.48 | 0.30 | 119.82 |
| **GAARM** | 54.52 | 37.53 | 0.30 | 100.18 |
| **GAIM16** | 45.04 | 36.59 | 0.30 | 100.18 |
| **DFA17** | 31.24 | 33.57 | 0.30 | 100.18 |

In some of the runs, GAAR02, GAARM, GAIM16 were also able to produce the best solution obtained by DFA17. However, the average costs of solutions by the algorithms were much higher at $59.85, $54.52 and $45.04 compared with $31.24 for DFA17. Similarly, the standard deviations of costs obtained by the algorithms were $38.48, $37.53 and $36.59 for GAAR02, GAARM, GAIM16 respectively. The worst solution produced by GAAR02 was as high as $119.82 whereas those of GAARM, GAIM16 and DFA17 had costs of $100.18 each.

For the case study, the results indicated that DFA17 had the ability to obtain reroutes with costs that were nearly half of those of GAAR02 on the average. This implied the improved search ability of DFA17 due to its enhanced movement function. Similarly, the reroutes produced by GAARM had better costs than those of GAAR02. In addition, the costs produced by GAIM16 were better than those of GAARM, most likely due to the proposed mutation strategy used by GAIM16. Overall, DFA17 had the most improved costs and GAIM16 had the second-best performance, in terms of route costs.

## 7.4 Scenario with multiple weather regions and aircraft

This scenario considers multiple weather regions with different weather intensities. Compared with the previous scenario with one weather region, this case had two additional weather regions of medium and high intensities. One of the weather regions was centred at a waypoint with longitude of 54.0953 and latitude of -1.9958. The second weather region was located around a waypoint with longitude and latitude of 53.6275 and -3.9453 respectively. The diameter of each weather regions was 6 km. The aircraft had a speed of 457 km/hr (flightaware.com). In addition, multiple aircraft were in the airspace under consideration (Fig. 7.5). The trajectories of these aircraft followed standard routes specified in the UK SRD document (NATS 2017) and are shown in the figure as black dashed lines.

The first aircraft was en route to London Gatwick Airport (EGKK) from Glasgow International Airport (EGPF) and had an average speed of 478 km/hr (flightaware.com). The second aircraft had a speed of 463 km/hr and was en route to Jersey Airport (EGJJ) from Manchester Airport (EGCC). The third aircraft was flying from Glasgow International Airport (EGPF) to Birmingham International

Airport (EGBB) and had a speed of 457 km/hr. The simulation parameters are as given in Table 7.1.

Fig. 7.5 shows the best alternative path generated by DFA17 for the scenario. DFA17 generated an alternative route that was able to avoid all the weather regions and the other aircraft. This path passes through waypoints TLA, IPDOR, ESKDO, INREV, UTOGU, INPIP, ABEVI, SHAPP, ERGAB, RIBEL, NELSA, POL, BARTN, LOVEL, LISTO and HON. The additional cost of the obtained reroute was $0.39 and was due to fuel cost associated with the longer path. Hence, the proposed algorithm was able to obtain an alternative path that kept additional costs to the minimum.

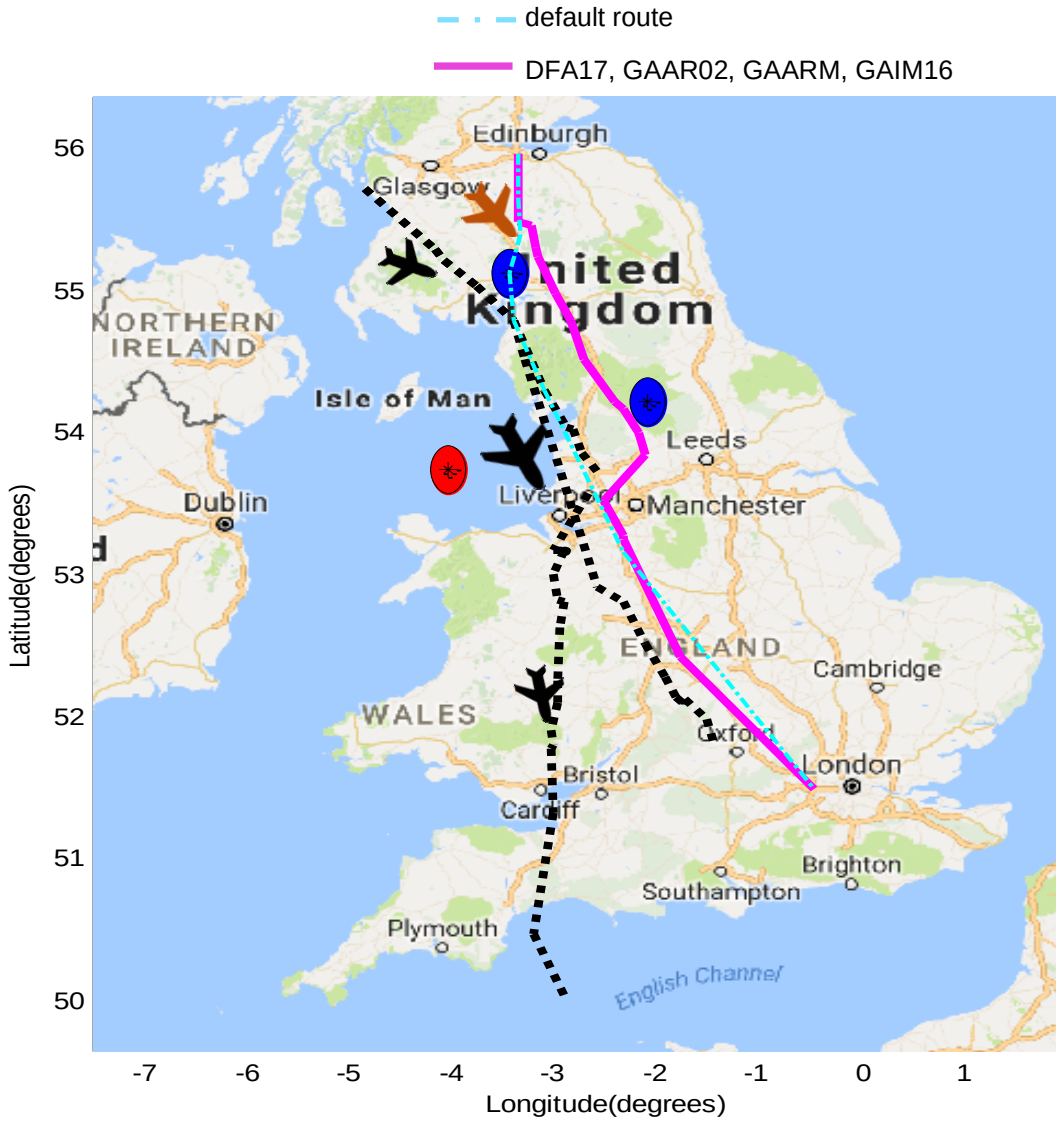


Fig. 7.5: Alternative flight path for scenario considering other aircraft in airspace and multiple weather regions

The obtained path was close to the leftmost weather region. This shows that the proposed algorithms are efficient in searching for solutions in complex scenarios. In situations where more weather separation is required, an increased buffer region can be added to the weather region input to the rerouting algorithms.

The cost of this alternative route was slightly higher than that generated for the previous scenario which had a single weather region. This is attributed to the increased complexity of this scenario with multiple aircraft and weather regions.

Similar observations are seen when average costs of solutions are considered From Table 7.3, the average cost for solutions obtained by DFA17 was $79.27. This value is more than twice the cost of $31.24 obtained for the scenario with a single weather region. This increase is likely due to the greater complexity of the current scenario.

Table 7.3: Details of results for multiple weather regions

| Algorithm | Average ($) | Standard deviation ($) | Best cost ($) | Worst cost ($) |
|---|---|---|---|---|
| **GAAR02** | 104.76 | 21.51 | 0.39 | 153.62 |
| **GAARM** | 100.03 | 24.17 | 0.39 | 133.38 |
| **GAIM16** | 99.42 | 20.37 | 0.39 | 146.55 |
| **DFA17** | 79.27 | 37.24 | 0.39 | 119.82 |

The other algorithms were able to obtain the same best path and cost. However, compared with DFA17, GAARM had a higher average value of $100.03. GAIM16 had the second-best performance in terms of average cost of $99.42. The worst average cost of $104.76 was obtained by GAAR02.

Compared with the scenario with one weather cell, GAAR02, GAARM and GAIM16 also produced solutions with much higher costs on the average. Overall, the results indicated that DFA17 had the best efficiency in obtaining the best solution. The second-best solution was by GAIM16 and the worst performance was by GAAR02.

The average processing time for this scenario was 1.8172s, 3.4784s, 5.1686s and 276.0178s for GAAR02, GAARM, GAIM16 and DFA17 respectively. It could be seen that DFA17 had the highest processing time. Whereas DFA17 produced solutions with the best costs, its improved search ability involved higher

processing time. Similar behaviour was observed for the scenario involving only one weather region (Section 7.3). In that scenario with one weather region, the processing time was 1.0100s, 4.1720s, 7.6060s and 429.6080s for GAAR02, GAARM, GAIM16 and DFA17. The timing results for the scenario (Section 7.3) were slightly higher for all the algorithms except GAAR02. This may be caused by the probabilistic nature of the algorithms. GAIM16 had the second-best processing time and the second-best route cost. Therefore, it had the best trade-off between the processing time and route cost. DFA17, because of its superior cost performance, could be used in cases where medium processing time is allowed.

## 7.5 Summary

This chapter presented a case study of flight path rerouting during adverse weather in UK airspace. The performance of the proposed rerouting algorithms for the case study were evaluated. The proposed algorithms were able to obtain alternative paths that avoided regions of adverse weather and other aircraft. With regards to average cost, GAAR02 had the worst performance. The best and second-best performances were by the proposed DFA17 and GAIM16 algorithms respectively. Results also showed that increased airspace complexity due to multiple weather regions and aircraft caused average cost of solutions to increase for all the considered algorithms.

# Chapter 8 : CONCLUSIONS AND FUTURE WORK

## 8.1 Conclusions

The adverse weather avoidance procedure is a complex process and involves a lot of inputs such as aeronautical, flight and weather data. Two improved algorithms, cost models, and a framework had been developed in this work for adverse weather avoidance for aircrafts. The proposed algorithms have improved capabilities to explore global solutions and reduced the impacts of the generated reroutes on air passengers. The developed model and framework can be applied to generate alternative paths in other situations where segments of airspace need to be avoided. Such contingencies include emergency airspace restrictions due to air shows, accidents and similar events. The following conclusions and contributions have been made.

### 8.1.1 Passenger-centric cost model for flight path reroutes during adverse weather

Prior research did not adequately consider the impact of path reroutes on passengers. This work had derived a cost model to determine the impacts of reroutes on passengers. The constituent factors considered by the model include flight delays, missed connections, weather impact and flight level changes. The developed model also considered the impact of flight path reroutes on operational costs such as fuel costs. Results showed that by using the developed model, alternative paths could be generated that adequately minimised the inconvenience of the adverse weather process to air passengers.

### 8.1.2 An improved genetic algorithm-based technique for flight path rerouting during adverse weather

A key component of the weather avoidance procedure is to find alternative path by using flight path rerouting process. To ensure that the process is efficient, an improved GA-based method has been proposed. The technique tackled the problem of premature convergence and locally optimal solutions by using a modified mutation process. The method entails using a random gene insertion and replacement strategy. Overall results showed that the proposed GA-based technique had better performances compared with existing GAs. This improved

performance was in terms of average costs, standard deviation and probability of optimality. This was especially so in complex scenarios involving multiple regions of adverse weather and multiple aircraft in the considered airspace.

### 8.1.3 Firefly algorithm-based technique for flight path rerouting

Firefly algorithms use swarm-based approaches for solution search. This work proposed a discrete firefly-based technique that is suitable for discrete network-based scenarios. To achieve this, an improved movement method for the algorithm was proposed. The movement method used a combined random insertion and replacement strategy to improve solution search. Results showed that the proposed technique obtained alternative paths that minimised the impact of adverse weather avoidance.

Overall, the DFA-based approach had better performance compared with the considered GA-based techniques. This superior performance was measured in terms of average costs, probability of optimality and standard deviation of obtained costs. However, the proposed DFA-based technique had longer processing time compared with the GA-based approach. The DFA-based technique could be used to derive very efficient reroutes in situations where medium processing time is tolerated, such as in pre-departure flight rerouting.

### 8.1.4 Integrated data framework for in-flight adverse weather avoidance

For efficient flight rerouting, a lot of data sources need to be considered. Such relevant data includes aeronautical, flight and weather data. This work aimed to improve the efficiency of the weather rerouting process by integrating the relevant data sources in the developed framework. The framework integrated the sources by supporting middleware-based data access. In addition, by using the middleware-based approach, this framework supported the SWIM global air transport data infrastructure.

### 8.1.5 Simulation platform for flight path rerouting during adverse weather

There was a lack of suitable open research tools for rapid design and testing of adverse weather avoidance techniques. To tackle this challenge, this work developed a simulation platform that enables the preliminary design and evaluation of flight path reroutes. The goal was to improve the speed and ease

of developing and testing adverse weather avoidance techniques and impact. The architecture of the simulation environment was modular so that its constituent parts can be modified and upgraded independently. The platform was developed in MATLAB programming language. The platform allowed the visualisation of reroutes and saving to file of results for further analyses.

## 8.2 Future Work

The developed rerouting model can be enhanced by further considering aircraft performance characteristics. These characteristics include the speed at which an aircraft has the most efficient performance. Also, control constraints of the aircraft, such as maximum turn ratio, could be included in the model. This would ensure that aircraft are able to more closely fly paths proposed by the rerouting algorithms.

The proposed algorithms in this work assumed that generated alternative paths do not violate crew workloads. In this case, alternative paths need to be checked after generation for the crew workload requirement independently. For example, increased flight time should not violate the maximum number of hours pilots can work. Future work can consider the crew requirement to improve the proposed rerouting algorithms.

It would be of interest to further consider the impact of the rerouting process on sector capacities. This would ensure that the rerouting process does not adversely increase air traffic control workload. In addition, it would reduce sector congestion that could increase flight delays.

Further testing would be desirable to evaluate the performance of the proposed algorithms in situations involving weather in motion. This could improve efficiency by ensuring that airspace is put into use as soon as possible following the resolution of adverse weather.

To ensure ease of use and interaction with the developed simulation platform for less experienced users, a graphical user interface could be developed. This could complement the existing approach using scenario files and improve ease of use of the rerouting platform.

# References


Ahn, C. W. and Ramakrishna, R. S. (2002) A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on Evolutionary Computation* 6 (6), 566-579.

Alam, S., Bui, L. T., Abbass, H. A. and Barlow, M. (2006) Pareto Meta-heuristics for Generating Safe Flight Trajectories Under Weather Hazards. *Simulated Evolution and Learning, 6th International Conference.* 2006. Berlin, Heidelberg: Springer.

Arıkan, U., Gürel, S. and Aktürk, M. S. (2016) Integrated aircraft and passenger recovery with cruise ttime controllability. *Annals of Operations Research* 236 (2), 295-317.

Ayo, B. S. (2017) An improved genetic algorithm for flight path re-routes with reduced passenger impact. *Journal of Computer and Communications* 5 (07), 65-75.

Ayo, B. S., Hu, Y. F. and Li, J. p. (2017) Adverse weather avoidance considering flight level changes. *2017 Seventh International Conference on Innovative Computing Technology (INTECH).* 16-18 Aug. 2017.

Beauducel, F. (2012) *Greatcircle and loxodrome: "As the crow flies" and rhumb line path, distance and bearing (version 1.3).* MATLAB Central.

Bermudez, L., Cook, T., Forrest, D., Bogden, P., Galvarino, C., Bridger, E., Creager, G. and Graybeal, J. (2009) Web feature service (WFS) and sensor observation service (SOS) comparison to publish time series data. *International Symposium on Collaborative Technologies and Systems.* IEEE.

BSI (2008) *Geographic information — Temporal schema (BS EN ISO 19108:2005 incorporating corrigendum October 2006)*. BSI - British Standard.

BSI (2009) *Geographic information — Geography Markup Language (GML) (ISO 19136:2007)*. BSI - British Standards.

BSI (2013) *Geographic information — Observations and measurements (ISO 19156:2011)*. BSI Standards Limited

BSI (2015) *BS ISO 19103:2015, Geographic information — Conceptual schema language*. The British Standards Institution.

CAA (2015) *SERA Implementation: Key UK changes.* Civil Aviation Authority.

Cauchi, N., Theuma, K., Zammit, C., Gauci, J. and Zammit-Mangion, D. (2015) A decision support tool for weather and terrain avoidance during departure. *AIAA/IEEE Digital Avionics Systems Conference - Proceedings.*

Chaimatanan, S., Delahaye, D. and Mongeau, M. (2012) A methodology for strategic planning of aircraft trajectories using simulated annealing. In *ISIATM 2012, 1st International Conference on Interdisciplinary Science for Air traffic Management.*

Chaimatanan, S., Delahaye, D. and Mongeau, M. (2014) A Hybrid Metaheuristic Optimization Algorithm for Strategic Planning of 4D Aircraft Trajectories at the Continental Scale. *IEEE Computational Intelligence Magazine* 9 (4), 46-61.

Chen, S. and Greenfield, P. (2004) QoS evaluation of JMS: an empirical approach. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences.* 2004.

Cheung, J. C. H. (2017) Flight planning: node-based trajectory prediction and turbulence avoidance. *Meteorological Applications* 25, 78–85.

Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. (2001) *Web Service Definition Language (WSDL).* W3C Note.

CICTT. (2013) *Phase of Flight: Definitions and Usage Notes.* CAST/ICAO Common Taxonomy Team (CICTT).

Crescenzo, D. D., Strano, A. and Trausmuth, G. (2010) System wide information management: the SWIM-SUIT prototype. *Integrated Communications Navigation and Surveillance Conference (ICNS).* 2010.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N. and Weerawarana, S. (2002) Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE* 6 (2), 86-93.

Daniels, T. S., Schaffner, P. R., Evans, E. T., Neece, R. T. and Young, S. D. (2012) Creating a realistic weather environment for motion-based piloted flight simulation. *IEEE/AIAA 31st Digital Avionics Systems Conference (DASC).* 2012.

Davis, C., Kimo, Y. J. and Duarte-Figueiredo, F. L. (2009) OGC web map service implementation challenges for mobile computers. *17th International Conference on Geoinformatics.* 2009.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6 (2), 182-197.

Dhief, I., Dougui, N. H., Delahaye, D. and Hamdi, N. (2017) A new trans-Atlantic route structure for strategic flight planning over the NAT airspace. *IEEE Congress on Evolutionary Computation (CEC).* 5-8 June 2017.

Dijkstra, E. W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik* 1 (1), 269-271.

Dorigo, M. and Gambardella, L. M. (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1 (1), 53-66.

ECMA (2013) *ECMA-404: The JSON data interchange format.* Geneva: Ecma International.

EEA (2016) *EMEP/EEA air pollutant emission inventory guidebook 2016.* European Environment Agency.

Erzberger, H., Lauderdale, T. A. and Chu, Y.-C. (2010) Automated conflict resolution, arrival management and weather avoidance for ATM. *27th Congress of International Council of the Aeronautical Sciences.* Nice, France.

Eugster, P. T., Felber, P. A., Guerraoui, R. and Kermarrec, A. M. (2003) The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)* 35 (2), 114-131.

EUROCONTROL (2013) *Severe weather risk management survey: Final report*

EUROCONTROL (2016) *Network Operations Report - Analysis - January 2016.* The European Organisation for the Safety of Air Navigation (EUROCONTROL) -.

FAA. (2013a) *Recommendations on Expanding the Use of Portable Electronic Devices During Flight.* Portable Electronic Devices Aviation Rulemaking Committee to the Federal Aviation Administration.

FAA. (2013b) *Thunderstorms. (AC No: 00-24C).* Federal Aviation Administration (FAA), U.S. Department of Transportation.

FAA. (2016a) *Aviation Weather. (AC No: 00-6B).* Federal Aviation Administration (FAA), U.S. Department of Transportation.

FAA. (2016b) *Aviation Weather Services. (AC No: 00-45H).* Transportation, U. S. D. O. Federal Aviation Administration ( FAA ).

Fadzli, S. A., Abdulkadir, S. I., Makhtar, M. and Jamal, A. A. (2015) Robotic Indoor Path Planning Using Dijkstra's Algorithm with Multi-Layer Dictionaries. *2015 2nd International Conference on Information Science and Security (ICISS).* 14-16 Dec. 2015.

Fang, Q. (2017) *JSONlab: a toolbox to encode/decode JSON files*. MATLAB Central.

Fernandes, J. L., Lopes, I. C., Rodrigues, J. J. P. C. and Ullah, S. (2013) Performance evaluation of RESTful web services and AMQP protocol. *Fifth International Conference on Ubiquitous and Future Networks (ICUFN).*

Fielding, R. T. and Taylor, R. N. (2002) Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology* 2 (2), 115-150.

FIXM (2012) FIXM Developer's Manual. SESAR Joint Undertaking/Federal Aviation Administration.

FIXM (2017a) *FIXM v4.1.0 Core Schema Documentation.* Airservices Australia, DSNA, EUROCONTROL, GCAA UAE, IATA, International Coordinating Council of Aerospace Industries Associations, JCAB, NATS Limited, NAV CANADA, SESAR Joint Undertaking & US FAA.

FIXM (2017b) *Flight Information Exchange Model Operational Data Description.*

Gleich, D. (2008) *MatlabBGL*. MATLAB Central.

Goblet, V., Fala, N. and Marais, K. (2015) Identifying Phases of Flight in General Aviation Operations. *15th AIAA Aviation Technology, Integration, and Operations Conference.*AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics.

Gong, C., McNally, D. and Lee, C. H. (2015) Dynamic arrival routes: A trajectory-based weather avoidance system for merging arrivals and metering. *15th AIAA Aviation Technology, Integration, and Operations Conference.*

Hapner, M., Burridge, R., Sharma, R., Fialli, J., Stout, K. and Deakin, a. N. (2013) *Java Message Service Version 2.0.* Oracle America, Inc.

Hauf, T., Sakiew, L. and Sauer, M. (2013) Adverse Weather Diversion Model DIVMET. *Journal of Aerospace Operations* 2 (3).

Houdebert, R. and Ayral, B. (2010a) Making SWIM interoperable between US and Europe. In *Integrated Communications Navigation and Surveillance Conference (ICNS).* 2010. 4-1.

Houdebert, R. and Ayral, B. (2010b) Making SWIM interoperable between US and Europe. In *Integrated Communications Navigation and Surveillance Conference (ICNS).* 2010. C4-1 -- C4-8.

Hupe, P., Hauf, T. and Rokitansky, C. H. (2014) Case study of adverse weather avoidance modelling. *SIDs 2014 - Proceedings of the SESAR Innovation Days.*

IATA (2018) *Jet Fuel Price Monitor.* International Air Transport Association (IATA).

ICAO. (2007a) *Annex 3 - Meteorological Service for International Air Navigation.* ICAO (International Civil Aviation Organization).

ICAO (2007b) *Guidance material on comparison of surveillance technologies (GMST).* International Civil Aviation Organization ( ICAO ), Asia and Pacific.

ICAO. (2016a) *Guidelines for the Implementation of OPMET Data Exchange using IWXXM.*

ICAO (2016b) *Long-Term Traffic Forecasts Passenger and Cargo.* International Civil Aviation Organization.

ICAO (2017) *Manual on System Wide Information management (SWIM) concept.* Doc 10039. International Civil Aviation Organization (ICAO).

iMatix (2012) *Broker vs. Brokerless.*

Issarny, V., Caporuscio, M. and Georgantas, N. (2007) A perspective on the future of middleware-based software engineering. *Future of Software Engineering.* 2007.

ITU-T (2008) *Information technology – Abstract Syntax Notation One (ASN. 1): Specification of basic notation*. International Telecommunication Union.

Jia, Y., Bodanese, E., Phillips, C., Bigham, J. and Tao, a. R. (2014) Improved reliability of large scale publish/subscribe based MOMs using model checking. *IEEE Network Operations and Management Symposium (NOMS).* 2014.

Kai-quan, C., Yan-wu, T. and Wei, W. (2015) An evolutionary multi-objective approach for Network-wide Conflict-free Flight Trajectories Planning. In *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th.* 13-17 Sept. 2015. 1D2-1-1D2-10.

Karr, D., Vivona, R., Roscoe, D., Depascale, S. and Wing, D. (2012) Autonomous Operations Planner: A Flexible Platform for Research in Flight-Deck Support for Airborne Self-Separation. *12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.* Aviation Technology, Integration, and Operations (ATIO) Conferences. American Institute of Aeronautics and Astronautics.

Krozel, J., Lee, C. and Mitchell, J. S. B. (2006) Turn-Constrained Route Planning for Avoiding Hazardous Weather. *Air Traffic Control Quarterly* 14 (2), 159-182.

Krozel, J. and Murphy Jr, J. T. (2007) Weather hazard requirements for NGATS aircraft. *Integrated Communications, Navigation and Surveillance Conference, 2007. ICNS'07.* IEEE.

Köksoy, O. and Yalcinoz, T. (2008) Robust design using Pareto type optimization: A genetic algorithm with arithmetic crossover. *Computers & Industrial Engineering* 55 (1), 208-218.

Lee, L. H., Lee, C. U. and Tan, Y. P. (2007) A multi-objective genetic algorithm for robust flight scheduling using simulation. *European Journal of Operational Research* 177 (3), 1948-1968.

Li, N., Du, Y. and Chen, G. (2013) Survey of cloud messaging push notification service. *International Conference on Information Science and Cloud Computing Companion (ISCC-C).* 2013.

Lim, W. and Zhong, Z. (2017) Re-Planning of Flight Routes Avoiding Convective Weather and the ``Three Areas''. *IEEE Transactions on Intelligent Transportation Systems*.

Liu, Y., Cheng, Y., Hu, Y. F., Pillai, P. and Esposito, a. V. (2011) Air-ground service integration for future aeronautical communication using SOA. *Integrated Communications, Navigation and Surveilance Conference (ICNS).* 2011.

Maeda, K. (2012) Performance evaluation of object serialization libraries in XML, JSON and binary formats. *Second International Conference on Digital*

*Information and Communication Technology and it's Applications (DICTAP).*
McNally, D., Sheth, K., Gong, C., Love, J., Lee, C. H., Sahlman, S. and Cheng, J. H. (2012) Dynamic Weather Routes: A weather avoidance system for near-term trajectory-based operations. *28th Congress of the International Council of the Aeronautical Sciences 2012, ICAS 2012.* Vol. 5.
Medjahed, B. (2008) Dissemination protocols for event-based service-oriented architectures. *IEEE Transactions on Services Computing* 1 (3), 155-168.
Menasce, D. (2005) MOM vs. RPC: communication models for distributed applications. *IEEE Internet Computing* 9 (2), 90-93.
Meng, J., Mei, S. and Yan, Z. (2009) RESTful Web services: A solution for distributed data integration. *International Conference on Computational Intelligence and Software Engineering (CiSE).* 2009.
MIT (2017) *Flight Information Exchange Model Modelling Best Practices.* MIT Lincoln Laboratory.
Mundy, D. and Chadwick, D. W. (2004) An XML alternative for performance and security: ASN.1. *IT Professional* 6 (1), 30-36.
Muñoz, P., Barrero, D. F. and R-Moreno, M. D. (2014) A Statistically Rigorous Analysis of 2D Path-Planning Algorithms. *The Computer Journal* 58 (11), 2876-2891.
NATS (2010) *The Effect of Thunderstorms and Associated Turbulence in Aircraft Operations.* NATS Ltd - UK Aeronautical Information Service.
NATS (2017) *The UK Integrated Aeronautical Information Package (IAIP).* NATS Aeronautical Information Service (AIS).
Ng, H. K., Grabbe, S. and Mukherjee, A. (2009) Design and Evaluation of a Dynamic Programming Flight Routing Algorithm Using the Convective Weather Avoidance Model. *AIAA Guidance, Navigation, and Control Conference.*Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics.
Nguyen, M.-H., Alam, S., Tang, J. and Abbass, H. (2007) Ants-inspired dynamic weather avoidance trajectories in a traffic constrained en route airspace. In *Proceedings of the 6th Eurocontrol Innovative Research Workshop and Exhibition (CARE INO III).* Bretigny sur Orge, France.
Niu, Z., Yang, C. and Zhang, a. Y. (2014) A design of cross-terminal web system based on JSON and REST. *5th IEEE International Conference on Software Engineering and Service Science (ICSESS).* 2014/06//.
OASIS (2012) *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0.* OASIS Standard.
OGC (2006) *OpenGIS Web Map Server Implementation Specification.* Open Geospatial Consortium, Inc.
OGC (2010) *OpenGIS Web Feature Service 2.0 Interface Standard.* OGC 09-025r1. Open Geospatial Consortium, Inc.
OGC (2012) *OWS-8 Aviation: Guidance for Retrieving AIXM 5.1 data via an OGC WFS 2.0, Engineering Report.* Open Geospatial Consortium, Inc.
OMG (2007) *Data Distribution Service for Real-time Systems Version 1.2.* Object Management Group (OMG).
OMG (2010) *The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification (DDS-RTPS).* Object Management Group (OMG).
OMG (2011) *Common Object Request Broker Architecture (CORBA) Specification, Version 3.2 - Part 2: CORBA Interoperability.* Object Management Group (OMG).

OMG (2014) *DDS Security. OMG Adopted Beta Specification.* Object Management Group (OMG).

Osaba, E., Yang, X.-S., Diaz, F., Onieva, E., Masegosa, A. D. and Perallos, A. (2017) A discrete firefly algorithm to solve a rich vehicle routing problem modelling a newspaper distribution system with recycling policy. *Soft Computing* 21 (18), 5295-5308.

O'Hara, J. (2007) Toward a Commodity Enterprise Middleware. *Queue* 5 (4), 48-55.

Papazoglou, P. M. and van den Heuvel, W.-J. (2007) Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal* 16 (3), 389-415.

Park, Y. and O'Kelly, M. E. (2014) Fuel burn rates of commercial passenger aircraft: variations by seat configuration and stage distance. *Journal of Transport Geography* 41, 137-147.

Ping, T. Y. (2003) *Multi-objective genetic algorithm for robust flight scheduling*. MEng Thesis. National University of Singapore.

Qing, G., Zheng, Z. and Yue, X. (2017) Path-planning of automated guided vehicle based on improved Dijkstra algorithm. *2017 29th Chinese Control And Decision Conference (CCDC).* 28-30 May 2017.

Roshen, W. (2009) *SOA-Based Enterprise Integration: A Step-by-Step Guide to Services-based Application.* 1 edition. New York: McGraw-Hill Education.

Roy, K. and Tomlin, C. J. (2007) Solving the aircraft routing problem using network flow algorithms. *American Control Conference, 2007. ACC '07.* 9-13 July 2007.

Sauer, M., Steiner, M., Sharman, R. D., Pinto, J. O. and Hauf, T. (2017a) Flight execution and route adaptation considering multiple constraints. *17th AIAA Aviation Technology, Integration, and Operations Conference, 2017.*

Sauer, M., Steiner, M., Sharman, R. D., Pinto, J. O. and Hauf, T. (2017b) Flight Execution and Route Adaptation Considering Multiple Weather Hazards. In *WMO Aeronautical Meteorology Scientific Conference.* Toulouse, France.

Schmidt, D. C. (2002) Middleware for real-time and embedded systems. *Communications of the ACM* 45 (6), 43-48.

Schmidt, D. C. and Buschmann, F. (2003) Patterns, frameworks, and middleware: their synergistic relationships. *25th International Conference on Software Engineering.* 2003.

Schneider, S. (2010) *What Is Real-Time SOA?* Real-Time Innovations, Inc.

Serme, G., de Oliveira, A. S., Massiera, J. and Roudier, a. Y. (2012) Enabling message security for RESTful services. *IEEE 19th International Conference on Web Services (ICWS).*

Sermi, F., Cuccoli, F., Mugnai, C. and Facheris, L. (2015) Aircraft hazard evaluation for critical weather avoidance. In *2015 IEEE Metrology for Aerospace (MetroAeroSpace).* 454-459.

SESAR JU (2012) *FIXM v1.1 Primer.* U.S. FAA & SESAR Joint Undertaking.

SESAR JU (2016) *SESAR SWIM Factsheet: System Wide Information Management (SWIM).*

Shahnaz, A., Nafees, T. and Azam, a. F. (2012) Domain based analysis of messaging patterns in service-oriented architecture. *5th International Conference on Biomedical Engineering and Informatics (BMEI).* 2012.

Standley, J., Brown, V., Comitz, P. and Schoolfield, a. J. (2012) SWIM segment 2 deployment and utilization in NextGen R&D programs. *Integrated Communications, Navigation and Surveillance Conference (ICNS).* 2012.

Stewart, T., Askey, L. and Hokit, M. (2012) A Concept for Tactical Reroute Generation, Evaluation and Coordination. *12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.*Aviation Technology, Integration, and Operations (ATIO) Conferences. American Institute of Aeronautics and Astronautics.

Strohmeier, M., Schäfer, M., Lenders, V. and Martinovic, a. I. (2014) Realities and challenges of nextgen air traffic management: the case of ADS-B. *IEEE Communications Magazine* 52 (5), 111-118.

Subramoni, H., Marsh, G., Narravula, S., Lai, P. and Panda, D. K. (2008) Design and evaluation of benchmarks for financial applications using Advanced Message Queuing Protocol (AMQP) over InfiniBand. In *Workshop on High Performance Computational Finance (WHPCF).* 2008. 1-8.

Taylor, C., Liu, S., Larsen, D., Wanke, C. and Stewart, T. (2017a) Designing flight specific reroutes using network optimization. *17th AIAA Aviation Technology, Integration, and Operations Conference, 2017.*

Taylor, C., Liu, S., Wanke, C. and Stewart, T. (2017b) Generating diverse reroutes for tactical constraint avoidance. *12th USA/Europe Air Traffic Management R and D Seminar.*

Taylor, C. and Wanke, C. (2009) Dynamic generation of operationally acceptable reroutes. In *9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO).* 2009.

Taylor, C. and Wanke, C. (2010) Generating Operationally-Acceptable Reroutes Using Simulated Annealing. *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference.*Aviation Technology, Integration, and Operations (ATIO) Conferences. American Institute of Aeronautics and Astronautics.

Upadhyaya, B., Zou, Y., Xiao, H., Ng, J. and Lau, a. A. (2011) Migration of SOAP-based services to RESTful services. *13th IEEE International Symposium on Web Systems Evolution (WSE).*

Veness, C. (2017) *Calculate distance, bearing and more between latitude/longitude points.* Movable Type Scripts.

Vinoski, S. (2006) Advanced Message Queuing Protocol. *IEEE Internet Computing* 10 (6), 87-89.

W3C, W. S. A. W. G. (2004) *Web Services Architecture, W3C Working Group Note.*

Wang, L. and Yang, H. (2013) Rerouting strategy research based on improved ant colony algorithm. *8th IEEE Conference on Industrial Electronics and Applications (ICIEA).* 19-21 June 2013.

Warren, I., Meads, A., Srirama, S., Weerasinghe, T. and Paniagua, a. C. (2014) Push notification mechanisms for pervasive smartphone applications. *Pervasive Computing, IEEE* 13 (2), 61-71.

Windhorst, R., Refai, M. and Karahan, S. (2009) Convective weather avoidance with uncertain weather forecasts. *AIAA/IEEE Digital Avionics Systems Conference - Proceedings.*

WMO (2014) *Guidelines on data modelling for WMO codes, Commission for Basic Systems Extraordinary Session, Asunción.* Geneva, Switzerland:

WMO. (2016a) *ICAO Meteorological Information Exchange Model.*

WMO (2016b) *Technical Regulations, Basic Documents No. 2, Volume II – Meteorological Service for International Air Navigation.* Geneva: World Meteorological Organization.

Wu, X., Feng, Z., Zhu, J. and Allen, R. (2007) GA-based path planning for multiple AUVs. *International Journal of Control* 80 (7), 1180-1185.

Xie, Z. and Zhong, Z. W. (2016) Aircraft Path Planning under Adverse Weather Conditions. *MATEC Web of Conferences* 77, 15001.

Xu, X., Li, C. and Zhao, Y. (2010) Air traffic rerouting planning based on the improved artificial potential field model. In *Chinese Control and Decision Conference (CCDC).* 26-28 May 2010. 1444-1449.

Yang, X.-S. (2009) Firefly algorithms for multimodal optimization. *Stochastic Algorithms: Foundations and Applications - 5th International Symposium.* Sapporo, Japan, 2009. Springer Berlin Heidelberg.

Zhang, J., Wu, J. and Song, T. (2014) Study of multi-aircraft conflict resolution and algorithm optimization based on genetic algorithm. *International Conference on Computer, Communications and Information Technology.* 2014. Atlantis Press.

Zhang, W., Kamgarpour, M., Sun, D. and Tomlin, C. J. (2012) A Hierarchical Flight Planning Framework for Air Traffic Management. *Proceedings of the IEEE* 100 (1), 179-194.

Zhiyang, L. and Tao, J. (2017) Route planning based on improved artificial potential field method. *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS).* 16-18 June 2017.

# Appendix I

SOFTWARE CODE

## A. Scenario file

function gen_ahn_ga_dfa1b

```
%by B.S. AYO
% last update: 13/04/18

close all
clear all
profile on

%load network data
load('ahnntwk.mat', 'ahnntwk'); %spp of 142 from node 1 to 20
load('ahn_coord.mat', 'ahn_coord'); %spp of 142 from node 1 to 20
%specify type of simulation
test_type = 'num_gen_test'; x_label = 'number of generations';

%provide values of parameters
step_gen = 5; max_gen_mult = 20;
num_runs = 200;  %number of times to run each algorithm

% max_num_gen = 20;
num_of_pop = 20;
num_of_pop1 = num_of_pop;num_of_pop2 = num_of_pop;
num_of_pop3 = num_of_pop;

mut_prob = 0.1;
mut_prob1 = mut_prob;mut_prob2 = mut_prob;mut_prob3 = mut_prob;

cros_prob = 0.9;
cros_prob1 = cros_prob;cros_prob2 = cros_prob;cros_prob3 = cros_prob;

num_of_ffly = 10; %number of fireflies

%specify algorithms and their parameters
%%% FIRST ALGORITHM
algo1_mut = 'mut_ahn'; algo1_label = 'GAAR02';
algo1 = @sga_rt3D_5b;
%%% SECOND ALGORITHM
algo2_mut = 'mut_swap'; algo2_label = 'GAARM';
algo2 = @sga_rt3D_5b;
%%% THIRD ALGORITHM
algo3_mut = 'mut_prop'; algo3_label = 'GAIM16';  %GAIM16
algo3 = @sga_rt3D_5b;
%%% FOURTH ALGORITHM (Not for GA-based)
algo4_mut = 'mut_prop'; algo4_label = 'DFA17';
algo4 = @dfa_rt3D_3a;
num_of_pop4 = num_of_ffly;
cros_prob4 = 0.95; %for DFA
mut_prob4 = 1; %for DFA
algo4_enabled=true;
%%%%%%

%create folder for results
simul_name = 'res_ahn_ga_dfa_';
folder_name = strcat(simul_name,test_type,datestr(now, 'yymmddHHMMSS'));
```

```
mkdir(folder_name);
profilename = strcat(folder_name,'/test_profiles1a',datestr(now,'HHMMSS'));

burn_coeff = 1;
time_spacing2 = Inf; %for ahnnwtk
spd2 = 1;%for ahnntwk, neutralizing effect of multiplier

threeD = false; %for ahnntwk
level_changes = false;  %for ahnntwk
coll_enabled = false; %for ahnntwk
gmaps_enabled = false;
show_wth_regions = true;
draw_indiv_paths = true;
gen_of_interest = [1;ceil(max_gen_mult/2);max_gen_mult];
get_prob_optim = true;
optimal_val = 142;
pth_line_wgt = 3; %line thickness for paths
coord = ahn_coord;
adjMSample = ahnntwk;

show_weights = true;
show_nodes = true; node_colr = 'yellow'; link_colr = 'black';
disp_options{1,1} = node_colr; disp_options{2,1} = link_colr;
disp_options{3,1} = show_nodes; disp_options{4,1} = show_weights;

wdelta = 0; wm = 0; wMc = 0; wl = 0;wb= 1; %for ahnntwk
wgtg (1,1)= wdelta;wgtg (2,1)= wm; wgtg (3,1) = wMc;
wgtg (4,1) = wl; wgtg (5,1) = wb;

xycoord =  [coord, zeros(size(coord,1),1)]; %set z-coord as 0's
start_time = 0; exp_arr_time = 0;next_cxn_time = 0; %ignore timings for ahnntwk
exp_times = [start_time; exp_arr_time; next_cxn_time];
deft_path = {}; %for ahnntwk
deft_dist = 0; %for ahnntwk

start_wp = 1; dest_wp = 20;% for ahnntwk

%define weather intensity values
w_low = 20; w_medium = 50; w_high = 100;
%indicate weather-impacted regions
%format: wth coord, start time, end time, intensity
sep_wth = 0.1; %wth radius + min_wth_sep
wth_loc = []; %for no adverse weather

%load planned trajectories of aircraft in format "[time,wpt; "
traj= {}; traj_path{1,1}={};

%plot labels
% x_label='number of generations';
y_label= 'cost ($)';
algo_legend = {algo1_label; algo2_label;algo3_label;algo4_label};
lformat =  {'-.b*'; '--rx'; '- g+'; '- mo'};

%run actual simulation algorithm
test_3D_20180202a;
%save code profiling files
profile off
profsave(profile('info'),profilename);
```

## B. Rerouting algorithms

### i. Rerouting algorithm for DFA17

```
function [best_fitness, best_cost_all, best_cost_compo, best_cost_compo_all, ...
            mean1,mean_cost,best_path_all] = ...
            dfa_rt3D_3a (cand_route,adjMSampleCell, ...
            start_wp2, dest_wp2,num_of_pop2,max_num_gen,~,mut_type,gamma,
mut_prob,exp_times, wgtg)

%define parameters
%start_wp2=1; dest_wp2=20; max_num_gen = 50;mut_prob=0.5;
num_of_pop2=length(cand_route);
pop_path = {};
%initialize fireflies
%cand_route = g_init_pop_quo(start_wp2, dest_wp2, adjMSample, num_of_pop2);
coll_status=0;
best_cost_so_far = Inf;
best_cost = Inf(max_num_gen,1);
mean_cost = Inf(max_num_gen,1);
best_cost_compo = cell(max_num_gen,1);
best_cost_all = Inf(max_num_gen,1);
% gamma=0.95;

%initialize light intensities
pop_cand_costArr =
get_cost_route2(cand_route,adjMSampleCell,start_wp2,dest_wp2,exp_times, wgtg);
light_inten = sum (pop_cand_costArr, 2);

num_gen=1;
 while (num_gen < max_num_gen)

  for xi = 1:num_of_pop2
%       for xj = xi+1:num_of_pop2
      for xj = 1:xi
        if light_inten(xj)<light_inten(xi)
          %calculate Hamming distance
          Flyi = cand_route{xi};
          Flyj = cand_route{xj};
          lenj = size(Flyj,1); leni =size(Flyi,1);
        if lenj>0 && leni>0
          if lenj>=leni
           truncFlyj = Flyj(1:leni);
           rij = (lenj-leni)+ sum(Flyi~=truncFlyj)-1; %remove one for end node
          else
           truncFlyi= Flyi(1:lenj);
           rij = (leni-lenj)+ sum(Flyj~=truncFlyi)-1;
          end
        end
          %move ffly
          %gamma_r = ceil(rij*gamma^2);
          gamma_r = ceil(rij*gamma^num_gen);
          nMut = ceil(2+(gamma_r-2)* rand(1,1));
          mut_route = cell(nMut,1);
          for m = 1: nMut
           mut_route{m,1} = Flyi;
          end
          %do mutation
```

```
   switch mut_type
    case 'mut_prop'
        childn_mutM = mutate_rt_ins3a(mut_route,adjMSampleCell,mut_prob);
        childn_mutM = mutate_rt_swap3a(childn_mutM,adjMSampleCell,mut_prob);
        [childn_mutM coll_status]  = rpair_route2a(childn_mutM,start_wp2, dest_wp2,
adjMSampleCell,exp_times);
    case 'mut_ahn'
        childn_mutM = mutate_route_quo(mut_route,adjMSampleCell{1,1},mut_prob);
    case 'mut_swap'
        childn_mutM = mutate_rt_swap3a(mut_route,adjMSampleCell,mut_prob);
        [childn_mutM coll_status]  = rpair_route2a(childn_mutM,start_wp2,
dest_wp2,adjMSampleCell,exp_times);
    otherwise
        disp('mutation type is not recognised')
   end

          %get best fly after movement
          costij = get_cost_route2(childn_mutM,adjMSampleCell,start_wp2,dest_wp2,exp_times,
wgtg);
          light_intenij = sum (costij, 2);
          %Iij0=light_intenij(light_intenij~=0)
          [bc_mut, best_id_mut]= min (light_intenij);
          Flyi = childn_mutM {best_id_mut,1};
          %pause
          cand_route{xi} = Flyi;
          pop_cand_costArr (xi,:) = costij(best_id_mut,:);
          light_inten(xi) = bc_mut;
       end
     end
  end

% pop_cand_costArr =
get_cost_route2(cand_route,adjMSampleCell,start_wp2,dest_wp2,exp_times, wgtg);
% light_inten = sum (pop_cand_costArr, 2);
[bc, best_id]= min (light_inten);
best_cost (num_gen,1) = bc;
if bc < best_cost_so_far
   best_cost_so_far = bc; %pause
   best_path_all = cand_route{best_id};
   best_cost_compo_all = pop_cand_costArr(best_id, :);
end

best_cost_all (num_gen,1) = best_cost_so_far;
best_fitness = best_cost_so_far;

mean_cost (num_gen,1)= mean (light_inten);
mean1 = mean (mean_cost);

best_cost_compo {num_gen,1} = pop_cand_costArr(best_id, :);

num_gen = num_gen+1;
 end
```

### ii. Rerouting algorithm for GAAR02, GAARM and GAIM16

```
function [best_fitness, best_cost_all, best_cost_compo, best_cost_compo_all, ...
             mean1,mean_cost,best_path_all, coll_status] = ...
             sga_rt3D_5b (cand_route,adjMSampleCell,start_wp2, dest_wp2,...
             num_of_pop2,max_num_gen,select_type,mut_type,cros_prob, ...
             mut_prob, exp_times, wgtg)
```

```
num_gen=1;
best_cost = Inf(max_num_gen,1);
mean_cost = zeros(max_num_gen,1);
best_route = cell(max_num_gen,1);
best_cost_compo = cell(max_num_gen,1);
best_cost_all = Inf(max_num_gen,1);
%best_cost_all = [];
best_cost_so_far = Inf;
coll_status=0;
%zeros(max_num_gen,1);
% pop_paths = cell(max_num_gen,2);
best_path_all = [Inf,Inf]; best_cost_compo_all= [Inf,Inf];
%Start main loop
while (num_gen <= max_num_gen)
%calculate total fitness function

pop_cand_costArr =
get_cost_route2(cand_route,adjMSampleCell,start_wp2,dest_wp2,exp_times, wgtg);
% pop_cand_costArr = get_cost_route2(cand_route,adjMSampleCell,start_wp2,dest_wp2);

pop_cand_cost = sum (pop_cand_costArr, 2);
% pop_paths {num_gen,1} = cand_route;
% pop_paths {num_gen,2} = pop_cand_cost;

%%%
%bc= max (pop_cand_cost(pop_cand_cost~=Inf))
%[bc, best_id]= min (pop_cand_cost(pop_cand_cost~=0));
[bc, best_id]= min (pop_cand_cost);
best_cost (num_gen,1)=bc;
best_route {num_gen,1} = cand_route{best_id};
%store components of costs
%best_cost_compo {num_gen,1} = pop_cand_costArr(best_id, 1:3);
best_cost_compo {num_gen,1} = pop_cand_costArr(best_id, :);
mean_cost (num_gen,1)= mean (pop_cand_cost);
%rp_2 = rpair_route(cand_route);

%check if cost is less than all previous values
   if bc < best_cost_so_far
     best_cost_so_far = bc; %pause
     best_path_all = cand_route{best_id};
     best_cost_compo_all = pop_cand_costArr(best_id, :);
   end
%end
%%%%%%%
best_cost_all (num_gen,1) = best_cost_so_far;

pop_cand_fitness = 1./pop_cand_cost;

crs_route = selection_2a(cand_route, pop_cand_fitness,cros_prob,select_type);


%child_route2 = rpair_route2_quo(crs_route,adjMSample);
%[child_route2,~] = rpair_route2(crs_route,adjMSampleCell, ...
%               start_wp2, dest_wp2, start_time);

child_route2 = crs_route;

% mutation if probability is met
switch mut_type
   case 'mut_prop'
```

```
        %childn_mutM = mutate_route_2a(child_route2,adjMSampleCell,mut_prob);
        childn_mutM = mutate_rt_ins3a(child_route2,adjMSampleCell,mut_prob);
        childn_mutM = mutate_rt_swap3a(childn_mutM,adjMSampleCell,mut_prob);
        [childn_mutM coll_status]  = rpair_route2a(childn_mutM,start_wp2, dest_wp2,
adjMSampleCell,exp_times);
    case 'mut_ahn'
        child_route2 = rpair_route2_quo(crs_route,adjMSampleCell{1,1});
        childn_mutM = mutate_route_quo(child_route2,adjMSampleCell{1,1},mut_prob);
    case 'mut_swap'
        childn_mutM = mutate_rt_swap3a(child_route2,adjMSampleCell,mut_prob);
        [childn_mutM coll_status]  = rpair_route2a(childn_mutM,start_wp2,
dest_wp2,adjMSampleCell,exp_times);
    otherwise
        disp('mutation type is not recognised')

end

child_route2 = childn_mutM;
%replace parent population with children
cand_route = child_route2;

num_gen = num_gen+1;
end
best_fitness = best_cost_so_far;
%best_fitness = min(best_cost_all);
mean1 = mean (mean_cost);
%best_path_all
```

## C. Mutation and repair functions

### i. GAAR02

```
function childn_mut = mutate_route_quo(this_route, adjMSampleMut, mut_prob)
childn_mut = cell(length(this_route),1);

%this_route
for k  = 1: length(this_route)
%mutate route
    parentMut = this_route{k,1};

    if length (parentMut)>2&& rand <= mut_prob
         start_wp = parentMut(1,1);
        dest_wp = parentMut(length(parentMut),1);
        %get random locus as mutation point
        rand_locus = ceil(rand*(length(parentMut)-2)+1);
        %remove all wp before mutation point from topology
     %for i = 1: length(parentMut)
     for i = 1: rand_locus-1
        wp_i = parentMut(i,1);
        for m = 1:length(adjMSampleMut)
        adjMSampleMut (m, wp_i) =0;
        adjMSampleMut (wp_i, m) =0;
        end
     end
        [partial_rt, ~] = get_cand_route_quo(parentMut(rand_locus),dest_wp,adjMSampleMut);
```

```
    %MI: check route generation was successful
    if  ~isempty(partial_rt)
      parentMut = [parentMut(1:rand_locus);partial_rt(2:length(partial_rt))];
    end

      childn_mut {k} = parentMut;
   else
      childn_mut {k} = parentMut;
   end
end

%%%%%%%%%
function repd = rpair_route2_quo (cand_route,adjMSample)
%repairs route by removing repeating waypoints
%[Ref: Ahn &Ramakrishna, 2002]
repd = cell(length(cand_route),1);

for i = 1: length(cand_route)
   d_route = cand_route {i,1};
   %check for and remove loops
   repd{i,1} = remov_loop(d_route);
end

function  this_rpd = remov_loop(this_route)
   this_rpd = this_route;
   lgtr = length(this_route);
   for n = 1: lgtr
     for m = 1: lgtr
     %for m = 1: length(this_route)
       if n<lgtr-m
      % if this_route(n) == this_route(m)&&n<m
          if this_route(n) == this_route(lgtr - m)
          this_rpd = [this_route(1:n);this_route((lgtr-m+1):lgtr)];
          end
       end
     end
   end
```

## ii. GAARM

```
%%%%%%
function childn_mut = mutate_rt_swap3a(this_route, adjMSampleCell, mut_prob)
adjMSample = adjMSampleCell{1,1};
childn_mut = cell(length(this_route),1);

%this_route
for k  = 1: length(this_route)
%%%%%%%%%%%%%%
%mutate route
   parentMut = this_route{k,1};
   if length (parentMut)>2&& rand <= mut_prob
      parentMut (ceil(rand*(length(parentMut)-2)+1),1)=ceil(rand*(length(adjMSample)));
      childn_mut {k} = parentMut;
       %mutd = parentMut'
   else
      childn_mut {k} = parentMut;
   end
end
```

## ii. GAIM16

```
%%%%%%
function childn_mut = mutate_rt_ins3a(this_route, adjMSampleCell, mut_prob)
adjMSample = adjMSampleCell{1,1};
childn_mut = cell(length(this_route),1);

%this_route
for k  = 1: length(this_route)
%mutate route
   parentMut = this_route{k,1};
   if length (parentMut)>2&& rand <= mut_prob
      rand_locus = ceil(rand*(length(parentMut)-2)+1);
      parentMut = [(parentMut (1:rand_locus))' ceil(rand*(length(adjMSample))) ...
          (parentMut ((rand_locus+1:length(parentMut))))'];
       childn_mut {k} = parentMut;
       %mutd = parentMut'
   else
       childn_mut {k} = parentMut;
   end
end

 %%%%%%

function childn_mut = mutate_rt_swap3a(this_route, adjMSampleCell, mut_prob)
adjMSample = adjMSampleCell{1,1};
childn_mut = cell(length(this_route),1);

%this_route
for k  = 1: length(this_route)
%%%%%%%%%%%%%
%mutate route
   parentMut = this_route{k,1};
   if length (parentMut)>2&& rand <= mut_prob
      parentMut (ceil(rand*(length(parentMut)-2)+1),1)=ceil(rand*(length(adjMSample)));
      childn_mut {k} = parentMut;
       %mutd = parentMut'
   else
       childn_mut {k} = parentMut;
   end
end


%%%%%%%%%%
function [repd, coll_detected] = rpair_route2a (cand_route,start_wp, end_wp,
adjMSampleCell2,exp_times)
%disp('running repair!!!!!!!!!!!!!!!')
%, start_wp, dest_wp,start_time)
%function y=rpair_route()
%e.g of input:
%cand_route = cell(4,1)
coll_detected = false;
adjMSample = adjMSampleCell2{1,1};
% adjMSampleCell{6,1} = {coll_enabled;threeD;spd2;time_spacing};
param_values = adjMSampleCell2{6,1};
coll_enabled = param_values(1);
threeD = param_values(2);
spd = param_values(3);
time_spacing = param_values(4);
% time_spacing = 5/60; %for 5 minutes
start_time = exp_times(1,1);
```

```
wpt_status = adjMSampleCell2{3,1};

max_attempt2 = 4;
repd = cell(length(cand_route),1);
%loop_exist = true;

for i = 1: length(cand_route)
    d_route = cand_route {i,1};
    last_id = length(d_route);
    loop_exist = true;
    lp_rmv = d_route;
    attempts = 1;
    attempts2= 1;
    discon = true; %first assume graph is disconnected
    while (discon)&& attempts2 <= max_attempt2
        attempts2= attempts2 +1;
    %%%%
    if (coll_enabled)
    [lp_rmv1, coll_status, adjMSample3] = rep_collision1a(lp_rmv, start_time, adjMSample,...
                        wpt_status, time_spacing, start_wp, end_wp, spd);
    coll_detected = coll_status;
    lp_rmv = lp_rmv1;
%%%
   adjMSample = adjMSample3;
    end
    %reconnect disjoint waypoints
    [lp_rmv2, discon] = recnt (lp_rmv,start_wp, end_wp, adjMSample);
     lp_rmv=lp_rmv2;
    %%
    end

    while (loop_exist == true) && attempts < 10
        %try to reconnect disjointed nodes
%       [lp_rmv, loop_exist] = remov_loop(d_route);
      [lp_rmv3, loop_exist] = remov_loop(lp_rmv);
       lp_rmv = lp_rmv3 ;

      %remove_loop_r =  lp_rmv
       attempts = attempts+1;
    end

    repd {i} = lp_rmv;
    %reset network data
    adjMSample = adjMSampleCell2{1,1};
end

function [recnt_r, discon]= recnt(this_route, start_wp, end_wp, adjMSample)
       recnt_r = this_route;
    %detect disconnection in graph
    discon = false;
    dsc_wp = cell(length(this_route),1); pos_dsc (1) = 0; dsc_id = 1;
 for t = 1: (length(this_route)-1)
   if adjMSample (this_route(t), this_route(t+1)) == 0  && ~isempty(this_route)
        discon = true;
        rt_con = g_init_pop_quo2a(this_route(t), this_route(t+1), adjMSample,1,10);

        rep_seg = rt_con{1,1};
        if size(rep_seg,1)>1
            dsc_id= dsc_id+1;
            pos_dsc(dsc_id) = t;
            dsc_wp(dsc_id,1) = {rep_seg};
```

```
        else
            break
        end%limit number of attempts to length of route
        if dsc_id > length(this_route)
            break
        end
    end
 end

if discon && size (dsc_wp,1)>1
    dsc_wp = dsc_wp(1:dsc_id);

for m = 2:length(dsc_wp) %ignore first entry used for initialisation
    item_m = dsc_wp{m,1};
    pos_m = pos_dsc(m);
    len_r = size(recnt_r,1);
    len_i = size(item_m,1);
    if size (item_m,1)>1
        recnt_r = [recnt_r(1:pos_m-1); item_m(1:len_i-1); recnt_r(pos_m+1:len_r)];
    end
    for n = m:length(pos_dsc)
        mlen =  length (item_m);
        pos_dsc(n) = pos_dsc(n) + len_i-2;
    end

end
end

 function  [this_rpd, loop_exist] = remov_loop(this_route)
    %this_rpd = this_route;
    loop_exist = false;
    for n = 1: length(this_route)
      for m = 1: length(this_route)
        %    if n<length(this_route)&&n<length(this_route)
        if this_route(n) == this_route(m)&&n<m
            %this_route(1:n);
           % this_route(m+1:length(this_route));
            this_route = [this_route(1:n);this_route(m+1:length(this_route))] ;
            loop_exist = true;
            break;
        end%reset loop
      end
      %stop - indices have changed
      if (loop_exist == true)
         break;
      end
    end
 this_rpd = this_route;

%%%%%%%%

%checks if collision is in view and gets alternative path
function  [this_rpd, collison_detected,adjMSample2] = rep_collision1a(this_route,...
                    start_time2, adjMSample2, wpt_status, time_spacing,...
                    start_wp, dest_wp, spd3)
collison_detected = false;
c=0;
coll_detected ={}; coll_wpt=[];
coll_wpt_ids =[];
this_rpd = this_route;
```

```
%get trajectory of aircraft ai1
ai1 = get_est_time_wpt (this_route, adjMSample2, start_time2, spd3);
if ~isempty(ai1)
%check no collision within each waypoint
 for j = 1: length(this_route)
   this_wpt_noted = false;
   wp_a = ai1(j,2);
   existing_slot = wpt_status {wp_a,1};
  for k = 1: length(existing_slot),
   % if ai1(j,1) == existing_slot(1,k),
   time_at_wpt = ai1(j,1);
   time_of_other= existing_slot(1,k);
   %check no other aircraft at interval -+time spacing
    if abs(time_of_other - time_at_wpt)<time_spacing
       collison_detected = true;
       if this_wpt_noted == false
         coll_wpt_ids = [coll_wpt_ids;j];
         this_wpt_noted = true;
       end
         c=c+1;
    end
  end

end

  del_col_rt = [];
  len = length(coll_wpt_ids);
  for m = 1: len
     wp_pos = coll_wpt_ids(m);
   if wp_pos >1
     wpt_id =  this_route(wp_pos);
   %delete affected waypoint from route if not start or destination
%       if wpt_id ~= start_wp && wpt_id ~= dest_wp
        this_route = [this_route(1:wp_pos-1); this_route(wp_pos+1:length(this_route))];
%           del_col_rt = [del_col_rt;this_route(wp_pos+1:length(this_route))];

       %eliminate waypoints from adjMSample
       for l = 1:length(adjMSample2)
        adjMSample2 (l, wpt_id) =0;
        adjMSample2 (wpt_id, l) =0;
       end
       %update positions of affected wpt
       coll_wpt_ids = coll_wpt_ids-1;
    end
   end

else
   collison_detected = true; %still indicate sth is wrong
end
c=0;
this_rpd = this_route;

%%%%%%%%%%


%%%%%%%%
function [ai] = get_est_time_wpt(route_arr, adjMDelay, start_time, av_spd)
ai=[];
% if ~isempty(route_arr)
if size(route_arr,1)>=2
ai = zeros(length(route_arr), 2);
```

```
ai(1, 1:2)= [start_time, route_arr(1,1)];
time_taken = start_time;
for i = 2: length(route_arr)
   %get two wpt
   wpi = route_arr(i-1,1);  wpj = route_arr(i,1);
   time_taken = time_taken+adjMDelay(wpi,wpj)/av_spd;
   %get the time cost up to the waypoint
   ai(i, 1:2)= [time_taken, route_arr(i,1)];
end
end
%%%%%%%%%

function [ai] = time_wpt_coord(route_arr, coord2, start_time, av_spd,dist_type)
%route_arr = time_param {1,1};
%start_time = time_param {2,1}
%start_time = 1; route_arr = [4; 2; 3; 8; 9];
% using 1NM =  1852m Ref: SI at bipm.org
% av_spd = 400/1.852; %Convert 200 NM/hr to km/hr
ai=[];
% if ~isempty(route_arr)
if size(route_arr,1)>=2
ai = zeros(length(route_arr), 2);
ai(1, 1:2)= [start_time, route_arr(1,1)];
time_taken = start_time;
for i = 2: length(route_arr)
     %get two wpt
   wpi = route_arr(i-1,1);  wpj = route_arr(i,1);
   dist_xy = get_dist1a(wpi, wpj, coord2, dist_type);
   time_taken = time_taken+dist_xy/av_spd;
   %get the time cost up to the waypoint
   ai(i, 1:2)= [time_taken, route_arr(i,1)];
end
end
```

## D. Main simulation algorithm

```
% function test_3D_20180119a
% clear all
% status of each waypoint
st_wp = cell(length(adjMSample),1);
traj_exists = exist('traj','var');
if (coll_enabled)
   if traj_exists==0
%        disp('TRAJ DOES NOT EXIST!')
   traj = cell(length(traj_path),1);
   end

   for t = 1: length(traj_path)
      if traj_exists==0
        traj{t,1} = get_est_time_wpt (traj_path{t,1}, adjMSample, start_time, spd2);
      end
   end
  for i = 1: length(traj)
    %get trajectory of ith aircraft
     ai = traj{i,1};
    %update status of each waypoint
    for j = 2: length(ai)
      wp_a = ai(j,2);
      wpt_existing = st_wp{wp_a,1};
```

```
      st_wp {wp_a,1} = [wpt_existing, ai(j,1)];
    end
  end

end

%initialise weather matrix
adjMSampleM = zeros(length(adjMSample));

%define array representing moving weather
num_wpt =length (adjMSample);
adjMSampleMovW = cell(num_wpt, 2);
adjMSampleMovW(1:num_wpt,1) = num2cell(1:num_wpt);

%detect and save wpts affected by weather
if size (wth_loc,1)>= 1  %if weather avoidance is enabled
   if threeD
      for w = 1: size (wth_loc,1)
         wth_centre = wth_loc (w, 1:3);
         x_wth =  wth_centre (1);
         y_wth =  wth_centre (2);
         z_wth =  wth_centre (3);
         total_sep_wth2 = wth_loc (w, 4)+sep_wth;
         wth_xtics = wth_loc (w, 5:7); %wth times and intensity
         for wpt_id = 1: length (coord)
            coord_wpt = coord(wpt_id, 1:3);
            x_wpt=  coord_wpt (1,1);
            y_wpt =  coord_wpt (1,2);
            z_wpt =  coord_wpt (1,3);
            dist_m = sqrt((x_wpt-x_wth)^2 +(y_wpt-y_wth)^2 +(z_wpt-z_wth)^2);
            if dist_m <= total_sep_wth2
              adjMSampleMovW{wpt_id,2} = wth_xtics;
            end
         end
      end
   else   %for backward compatibility
      geo_dist_exists = exist('geo_dist','var');
      if geo_dist_exists==1 && geo_dist==true
         %note coordinates are inverted
          %Coordinates converted using Geodetics
         wth_loc_new = earth_wcell;
         coord_new = earth_xycoord;
      else
         wth_loc_new = wth_loc;
         coord_new = coord;
      end

      for w = 1: size (wth_loc,1)
         wth_centre = wth_loc_new (w, 1:2);
         total_sep_wth2 = wth_loc (w, 3)+sep_wth;
         wth_xtics = wth_loc (w, 4:6); %wth times and intensity
         x_wth =  wth_centre (1);
         y_wth =  wth_centre (2);
         for wpt_id = 1: length (coord_new)
            coord_wpt = coord_new(wpt_id, 1:2);
            x_wpt=  coord_wpt (1,1);
            y_wpt =  coord_wpt (1,2);
            dist_m = sqrt((x_wpt-x_wth)^2 + (y_wpt-y_wth)^2);
            if dist_m <= total_sep_wth2
              adjMSampleMovW{wpt_id,2} = wth_xtics;
            end
```

```
          end
        end
    end
end

%set the cost arrays
adjMSampleCell = cell(6,1);
%make matrix sparse
adjMSample = sparse(adjMSample);
% array for delay cost
adjMSampleCell{1,1} = adjMSample;
% array for weather cost
adjMSampleM = sparse(adjMSampleM);
adjMSampleCell{2,1} = adjMSampleM;

%array for planned routes
adjMSampleCell{3,1} = st_wp;
%array for flight levels
if level_changes == true
adjMSampleCell{4,1} = flevels;
end
%array for moving weather
adjMSampleCell{5,1} = adjMSampleMovW;
%array for some parameter values
adjMSampleCell{6,1} = [coll_enabled;threeD;spd2;time_spacing2;...
    burn_coeff;deft_dist];

best_run = Inf (max_gen_mult,4);
% best_run = zeros (num_runs,4);
cxn_cost = zeros (num_runs,4);
best_path_arr = cell (max_gen_mult,4);
best_cost_gen = cell (max_gen_mult,4);
best_cost_compo_arr= cell (max_gen_mult,4);
mean_cost_cell = cell (max_gen_mult,4);
wpt_timing = cell (max_gen_mult,4);
best_gen_num = Inf (max_gen_mult,num_runs,4);
path_weight = cell (max_gen_mult,num_runs,4);
results_wght = cell (max_gen_mult,num_runs,4);

for i=1: num_runs

 runn = i

  cand_route = g_init_pop_quo(start_wp, dest_wp, adjMSample, num_of_pop2);

    for gen = 1: max_gen_mult
      gen_mult = gen;
%       runn
     %vary parameter of interest
     switch (test_type)
        case 'num_gen_test'
          max_num_gen = gen*step_gen;
        case 'num_pop_test'
          num_of_pop = gen*step_gen;
          num_of_pop1 = num_of_pop;num_of_pop2 = num_of_pop;
          num_of_pop3 = num_of_pop;num_of_pop4 = num_of_pop;
        case 'mut_test'
          mut_prob = gen*step_gen;
          mut_prob1 = mut_prob;mut_prob2 = mut_prob;
          mut_prob3 = mut_prob;
        case 'cros_test'
```

```
        cros_prob = gen*step_gen;
        cros_prob1 = cros_prob;cros_prob2 = cros_prob;
        cros_prob3 = cros_prob;
    end

   %find best path
   %     max_num_gen = gen*step_gen;
         %matrices to hold results
         [best_fitness1, best_cost_all1, best_cost_compo1, ...
           best_cost_compo_all1,mean1, mean_cost_arr1, best_path_all1] = algo1 (cand_route,
adjMSampleCell, ...
              start_wp, dest_wp,num_of_pop1,max_num_gen,'roulette',algo1_mut,...
             cros_prob, mut_prob1,exp_times, wgtg);
         best_run(gen,1)= best_fitness1;
         best_path_arr {gen,1} = best_path_all1;
         best_cost_compo_arr{gen,1} = best_cost_compo_all1;
         mean_cost_cell{gen,1} = mean_cost_arr1;
         best_cost_gen {gen,1} = best_cost_all1;
         wpt_timing {gen,1} = get_est_time_wpt (best_path_all1, adjMSample, start_time, spd2);

   %%
       [best_fitness2, best_cost_all2, best_cost_compo2, ...
            best_cost_compo_all2,mean2, mean_cost_arr2, best_path_all2] = ...
            algo2 (cand_route, adjMSampleCell, ...
               start_wp, dest_wp,num_of_pop2,max_num_gen,'roulette',algo2_mut,...
              cros_prob, mut_prob2,exp_times, wgtg);
         best_run(gen,2)= best_fitness2;
         best_path_arr {gen,2} = best_path_all2;
         best_cost_compo_arr{gen,2} = best_cost_compo_all2;
         mean_cost_cell{gen,2} = mean_cost_arr2;
         best_cost_gen {gen,2} = best_cost_all2;
         wpt_timing {gen,2} = get_est_time_wpt (best_path_all2, adjMSample, start_time, spd2);

   %%%
       [best_fitness3, best_cost_all3, best_cost_compo3, ...
            best_cost_compo_all3,mean3, mean_cost_arr3, best_path_all3] = ...
             algo3 (cand_route,adjMSampleCell, ...
              start_wp,
dest_wp,num_of_pop3,max_num_gen,'roulette',algo3_mut,cros_prob,mut_prob3,exp_times,
wgtg);
         best_run(gen,3)= best_fitness3;
         best_path_arr {  gen,3} = best_path_all3;
         best_cost_compo_arr {  gen,3} = best_cost_compo_all3;
         mean_cost_cell {  gen,3} = mean_cost_arr3;
         best_cost_gen {  gen,3}   = best_cost_all3;
         wpt_timing {  gen,3}  = get_est_time_wpt (best_path_all3, adjMSample, start_time,
spd2);
   %%%%
      if algo4_enabled
       [best_fitness4, best_cost_all4, best_cost_compo4, ...
            best_cost_compo_all4,mean4, mean_cost_arr4, best_path_all4] = ...
            algo4 (cand_route, adjMSampleCell, ...
               start_wp, dest_wp,num_of_pop4,max_num_gen,'roulette',algo4_mut,...
              cros_prob4, mut_prob4,exp_times, wgtg);
         best_run(gen,4)= best_fitness4;
         best_path_arr {gen,4} = best_path_all4;
         best_cost_compo_arr{gen,4} = best_cost_compo_all4;
         mean_cost_cell{gen,4} = mean_cost_arr4;
         best_cost_gen {gen,4} = best_cost_all4;
         wpt_timing {gen,4} = get_est_time_wpt (best_path_all4, adjMSample, start_time, spd2);
      end
```

```
    end

    rt_cost_compo_deft = get_cost_route2(deft_path,adjMSampleCell, start_wp,...
                        dest_wp, exp_times, wgtg);
    Jdeft_total=  sum(rt_cost_compo_deft);
    pth_usual = deft_path';
    for a = 1:4
      best_gen_num(:,i,a) = best_run(:,a);
      path_weight(:,i,a) = best_path_arr (:,a);
      results_wght(:,i,a) = best_cost_compo_arr(:,a);
    end

end

info_file = strcat(folder_name,'/rawinfo_',datestr(now, 'HHMMSS'));
save (info_file);

%save simulation parameters to file:
res_file = strcat(folder_name,'/res_cmb',datestr(now, 'HHMMSS'),'.csv');
file_id1 = fopen(res_file,'a+');
fprintf(file_id1,'\n ===============');
fprintf(file_id1,'\n SIMULATION ID: %s\t SIMULATION TYPE: %s ',folder_name, test_type);
fprintf(file_id1,'\n Algorithms run: %s\t  %s\t  %s\t  %s\t',...
algo1_label, algo2_label,algo3_label,algo4_label);
fprintf(file_id1,'\n Number of runs: %8.2f\t', num_runs);
fprintf(file_id1,'\n Step size: %8.2f\t Number of step: %8.2f\t', ...
step_gen, max_gen_mult);

if algo4_enabled
fprintf(file_id1,'\n Population size: %8.2f\t %8.2f\t %8.2f\t %8.2f\t', ...
num_of_pop1, num_of_pop2,num_of_pop3,num_of_pop4);
fprintf(file_id1,'\n Mutation rate: %8.2f\t %8.2f\t %8.2f\t %8.2f\t', ...
mut_prob1,mut_prob2,mut_prob3,mut_prob4);
fprintf(file_id1,'\n Crossover rate: %8.2f\t %8.2f\t %8.2f\t %8.2f\t', ...
cros_prob1,cros_prob2,cros_prob3,cros_prob4);
else
fprintf(file_id1,'\n Population size: %8.2f\t %8.2f\t %8.2f\t ', ...
num_of_pop1, num_of_pop2,num_of_pop3);
fprintf(file_id1,'\n Mutation rate: %8.2f\t %8.2f\t %8.2f\t', ...
mut_prob1,mut_prob2,mut_prob3);
fprintf(file_id1,'\n Crossover rate: %8.2f\t %8.2f\t %8.2f\t ', ...
cros_prob1,cros_prob2,cros_prob3);
end
fclose(file_id1);


%get average solution
% ave_gen_num1 = mean(best_gen_num1);
len_gen_int = length(gen_of_interest)
siz_soln = size(best_gen_num,3);
ave_soln = zeros(siz_soln,1);
std_soln = zeros(siz_soln,1);
best_soln = zeros(siz_soln,1);
worst_soln = zeros(siz_soln,1);
prob_optim = zeros(siz_soln,1);
res_cmb = cell(8,len_gen_int);
best_compo = Inf(siz_soln,length(wgtg));
worst_compo = Inf(siz_soln,length(wgtg));

for g = 1:length(gen_of_interest)
```

```
geni = gen_of_interest(g,1);
for m = 1:size(best_gen_num,3)
   %m is number of algorithms
   soln = best_gen_num(:,:,m);
   gen_row =  soln(geni,:)';
   ave_soln(m) = mean(gen_row);
   std_soln(m) = std(gen_row);
   [best_soln(m),best_id(m)] = min(gen_row);
   [worst_soln(m),worst_id(m)] = max(gen_row);
   min_pth = path_weight{geni,best_id(m),m};
   max_path = path_weight{geni,worst_id(m),m};
   paths_best {m}= min_pth'; %best_path in all runs
   compo_b=results_wght{geni,best_id(m),m};
   if length(compo_b)>1
   best_compo(m,:) = results_wght{geni,best_id(m),m};
   end
   paths_worst {m} = max_path'; %worst_path in all runs
   compo_w = results_wght{geni,worst_id(m),m};
   if length(compo_w)>1
   worst_compo(m,:) = compo_w;
   end
   if get_prob_optim == true;
   optim_ids = gen_row==optimal_val;
   prob_optim(m) = (sum(optim_ids))/length(gen_row);
   end
end
%show combined results
tbl = [ave_soln,std_soln,best_soln,worst_soln,prob_optim];
gen_intr = geni*step_gen;
res_cmb {1,g} = gen_intr;
res_cmb {2,g} = tbl;
res_cmb {3,g} = best_id; %indices of best paths
res_cmb {4,g} = paths_best;
res_cmb {  5,g}  = worst_id;%indices of worst paths
res_cmb {6,g} = paths_worst;
res_cmb {7,g} = best_compo;
res_cmb {8,g} = worst_compo;

formatSpec ='%d ';
file_id1 = fopen(res_file,'a+');
fprintf(file_id1,'\n----------\n DETAILS ON %s = %8.4d: ',x_label, gen_intr);
fprintf(file_id1,'\n ----------');
fprintf(file_id1,'\n BEST ID: %d \t  WORST ID: %d ',best_id,worst_id);
fprintf(file_id1,'\n-------------------------------------------------');
fprintf(file_id1,'\n ave_soln \t  std_soln \t  best_soln \t worst_soln \t  prob_optim ');
fprintf(file_id1,'\n-------------------------------------------------');
for linem = 1:length(paths_best)
   fprintf(file_id1,'\n');
   fprintf(file_id1,'%8.4f \t',tbl(linem,:));
end
fprintf(file_id1,'\n-------------------------------------------------');

fprintf(file_id1,'\n BEST PATHS: ');
for linem = 1:length(paths_best)
   fprintf(file_id1,'\n');
   fprintf(file_id1,formatSpec,paths_best{linem}');
end
fprintf(file_id1,'\n WORST PATHS: ');
for linem = 1:length(paths_worst)
   fprintf(file_id1,'\n');
   fprintf(file_id1,formatSpec,paths_worst{linem}');
```

```
    end

    fprintf(file_id1,'\n-----------------------------------------------');
    fprintf(file_id1,'\n Weights of costs: \n');
    fprintf(file_id1,'%8.4f \t ',wgtg);
    fprintf(file_id1,'\n-----------------------------------------------');
    fprintf(file_id1,'\n Components of best costs: ');
    fprintf(file_id1,'\n-----------------------------------------------');
     for linem = 1: size(best_compo,1)
        fprintf(file_id1,'\n');
        fprintf(file_id1,'%8.4f \t', best_compo(linem,:));
    end
    fprintf(file_id1,'\n-----------------------------------------------');

     fprintf(file_id1,'\n-----------------------------------------------');
    fprintf(file_id1,'\n Components of worst costs: ');
    fprintf(file_id1,'\n-----------------------------------------------');
     for linem = 1: size(worst_compo,1)
        fprintf(file_id1,'\n');
        fprintf(file_id1,'%8.4f \t', worst_compo(linem,:));
    end
    fprintf(file_id1,'\n-----------------------------------------------');

    fclose(file_id1);
 end


%display stdev etc for specified values
disp('-----------------------------------')
gen_interest = res_cmb(1,:)
celldisp(res_cmb(2,:))
disp('-----------------------------------')

%get best path for last indicated generation
last_gen = length(gen_of_interest);
best_paths_last = res_cmb {4,last_gen};
worst_paths_last = res_cmb {6,last_gen};
best_compo_last = res_cmb {7,last_gen};
worst_compo_last = res_cmb {8,last_gen};
for p = 1 : 4
best_pth{p}= best_paths_last{p};
bestpath_label = convert_route_id_to_labels1a(best_paths_last{p},nav_and_fixes)'
p
bestpath_label{1,1}
worst_pth{p}= worst_paths_last{p}';
best_compo_p{p}= best_compo_last (p,:);
worst_compo_p{p}= worst_compo_last(p,:);
end

disp('BEST PATHS AT LAST SELECTED GEN:')
celldisp(best_pth)
disp('BEST COST COMPONENTS:')
celldisp(best_compo_p)
this_best_path = best_paths_last{p}
COSTCOMPO_2 =
get_cost_route2({this_best_path},adjMSampleCell,start_wp,dest_wp,exp_times, wgtg);


%plot average solution
hold off
xAxis = (1:max_gen_mult)*step_gen;
```

```
for m = 1:size(best_gen_num,3)
   soln_data = best_gen_num(:,:,m);
   ave_gen_num1 = mean(soln_data,2);
   ave_gen_cmb{m,1} = ave_gen_num1;
   plot (xAxis,ave_gen_num1,lformat{m,1});
   hold on
end
xlabel(x_label);
ylabel(y_label);
legend(algo_legend);
fig_file = strcat(folder_name,'/avg_gen1a_',datestr(now, 'HHMMSS'));
savefig (fig_file);
%also save figure in EMF format
saveas(gcf,strcat(fig_file,'.emf'));
hold off

%save average values
file_id1 = fopen(res_file,'a+');
fprintf(file_id1,'\n-----------------------------------------------');
fprintf(file_id1,'\n AVERAGE COSTS AT: \n');
fprintf(file_id1,'%8.4f \t',xAxis);
fprintf(file_id1,'\n-----------------------------------------------');
for linem = 1:size(ave_gen_cmb,1)
  fprintf(file_id1,'\n');
  fprintf(file_id1,'%8.4f \t',ave_gen_cmb{linem});
end
fprintf(file_id1,'\n-----------------------------------------------');
fclose(file_id1);

%show epicentre of adverse weather
if (threeD == true)
%first get 3D coord
xyz_coord = coord3D_from_levels(xycoord, flevels);
else
xyz_coord= xycoord;
end

results_file = strcat(folder_name,'/res_data',datestr(now, 'HHMMSS'));
save(results_file);

if show_wth_regions == true
      clf
      wth_loc_inv = wth_loc; % x and y coord not swapped yet
      for dc = 1: size(wth_loc_inv,1)
         if threeD
            %%%%%%%%%%Uncomment if no weather region
            scatter3 (wth_loc_inv(dc, 1),wth_loc_inv(dc, 2),wth_loc_inv(dc, 3),200, 'mo','fill');
            hold on
         else
         %multiplying distance accross = 2*radius
         geo_dist_exists = exist('geo_dist','var');
         if geo_dist_exists==1 && geo_dist==true
           dc_coord = [wth_loc_inv(dc, 2) wth_loc_inv(dc, 1)];
           %Note:degree equiv estimated using R = 6378.1370 km (GRS80)
           %Ref: geom2d-2017.08.31 (refell fn)
           draw_region1a(dc_coord(1,1:2), 2*wth_loc_inv(dc, 3)/6378.1370*360, wth_loc_inv(dc,
6));
         else
           dc_coord = wth_loc_inv(dc, 1:2);
           draw_region1a(dc_coord(1,1:2), 2*wth_loc_inv(dc, 3), wth_loc_inv(dc, 6));
         end
```

```
            hold on
            plot(dc_coord(1,1),dc_coord(1,2), 'k*');
            hold on
            end
        end
        hold off
end

%get display options
s_options(1)= draw_indiv_paths;
s_options(2)= show_wth_regions;
s_options(3)= coll_enabled;
s_options(4)= gmaps_enabled;
s_options(5)= threeD;

%set values of variables
display_param{1,1} = s_options;
display_param{2,1} = adjMSample;
display_param{3,1} = coord;
display_param{4,1} = traj_path;
display_param{5,1} = folder_name;
display_param{6,1} = pth_line_wgt;
display_param{7,1} = algo_legend;
display_param{8,1} = disp_options;
display_param{9,1} = deft_path;

%show resultant paths
display_network1a(best_pth,lformat,display_param);

geo_dist_exists = exist('geo_dist','var');
if geo_dist_exists==1 && geo_dist==true
    xlabel('Longitude(degrees)');
    ylabel('Latitude(degrees)');
end

fig_file = strcat(folder_name,'/paths1a_all',datestr(now, 'HHMMSS'));
savefig (fig_file);
saveas(gcf,strcat(fig_file,'.emf'));
if gmaps_enabled
    %adds Google Maps background
    %Ref:https://uk.mathworks.com/matlabcentral/fileexchange/...
    %27627-zoharby-plot-google-map
    plot_google_map
    %save figure
    fig_file = strcat(folder_name,'/paths1a_all_gmap',datestr(now, 'HHMMSS'));
    savefig (fig_file);
    %also save figure in EMF format
    saveas(gcf,strcat(fig_file,'.emf'));
end

%also save figure in EMF format
saveas(gcf,strcat(fig_file,'.emf'));
%show individaul ptah if needed
if draw_indiv_paths == true
    for m = 1:size(best_gen_num,3)
        clf
        %show adverse weather region
        if show_wth_regions == true
            wth_loc_inv = wth_loc; % x and y coord not swapped yet
            for dc = 1: size(wth_loc_inv,1)
                if threeD
```

```
            %%%%%%%%%%Uncomment if no weather region
            scatter3 (wth_loc_inv(dc, 1),wth_loc_inv(dc, 2),wth_loc_inv(dc, 3),200,'mo','fill');
            hold on
          else
           %multiplying distance accross = 2*radius
          geo_dist_exists = exist('geo_dist','var');
          if geo_dist_exists==1 && geo_dist==true
            dc_coord = [wth_loc_inv(dc, 2) wth_loc_inv(dc, 1)];
             draw_region1a(dc_coord(1,1:2), 2*wth_loc_inv(dc, 3)/6378.1370*360,
       wth_loc_inv(dc, 6));
          else
            dc_coord = wth_loc_inv(dc, 1:2);
            draw_region1a(dc_coord(1,1:2), 2*wth_loc_inv(dc, 3), wth_loc_inv(dc, 6));
          end
%             draw_region1a(dc_coord(1,1:2), 2*wth_loc_inv(dc, 3), wth_loc_inv(dc, 6));
          hold on
          plot(dc_coord(1,1),dc_coord(1,2), 'k*');
          hold on
          end
        end
      hold off
      end
      %draw actual figure
      this_lformat = lformat(m);
      algo_name = algo_legend(m);
      display_param{7,1} = algo_name;
      display_network1a(best_pth(m),this_lformat,display_param);

      %save figure
      fig_file = strcat(folder_name,'/path',algo_name{1},datestr(now, 'HHMMSS'));
      geo_dist_exists = exist('geo_dist','var');
      if geo_dist_exists==1 && geo_dist==true
        xlabel('Longitude(degrees)');
        ylabel('Latitude(degrees)');
      end
      savefig (fig_file);
      %also save figure in EMF format
      saveas(gcf,strcat(fig_file,'.emf'));

      if gmaps_enabled
      plot_google_map
      %save figure
      fig_file = strcat(folder_name,'/path_gmap',algo_name{1},datestr(now, 'HHMMSS'));
      savefig (fig_file);
      %also save figure in EMF format
      saveas(gcf,strcat(fig_file,'.emf'));
      end

   end
end
```

## E. Calculation of cost

```
%%%%%
function [cost_route3] = get_cost_route2(cand_route,...
                    adjMSampleCell, start_wp, dest_wp, exp_times, wgtg)
%function [fit_route,cost_cxn,cost_no_cxn] = get_cost_route2(cand_route,...
%                     adjMSampleCell, start_wp, dest_wp, cxn_m)
```

```
%calculate cost of path
%%%%%%%%%%%%%%%%
%create cell for cost too
%%%%%%%%%%%%%%%
%initialize array for route cost
%len_adjs = length(adjMSampleCell)-1
% cost_route3 = Inf(length (cand_route),length(adjMSampleCell)-1);
cost_route3 = Inf(length (cand_route),length(wgtg));

% array for delay cost
%adjMSampleD = adjMSampleCell{1,1};
% array for weather cost
%adjMSampleM = adjMSampleCell{2,1};
wp_id =1;cost_this_route=0;
missed_cxn_cost = 50;
min_cxn_time=30/60;%convert to hour
%min_cxn_time=0.5;%convert to hour
%define values of per unit costs
Cdelta = 2*60;%convert to per min ( 0.05 )
Cm=50; CMc =50;
level_chg_cost = 50;

%define weights of the costs
wdelta = wgtg (1,1);wm = wgtg (2,1); wMc = wgtg (3,1); w1 = wgtg (4,1);
wb = wgtg (5,1);
%get expected arrival/departure times
start_time = exp_times(1,1);
exp_arr_time = exp_times(2,1);
next_cxn_time = exp_times(3,1);
% adjMSampleCell{6,1} = [coll_enabled;threeD;spd2;time_spacing2;...
%     burn_coeff;deft_dist];
param_values = adjMSampleCell{6,1};
coll_enabled = param_values(1);
threeD = param_values(2);
spd2 = param_values(3);

%based on max speed of 9NM/min in Arikan(2016)&10NM sep from thunderstorm
% min_wea_sep = 0.02;
min_wea_sep = 10*1.852/spd2;

time_spacing2 = param_values(4);
burn_coeff2 =  param_values(5);
deft_dist2 =  param_values(6);
adjMg = adjMSampleCell{1,1};
wpt_status2 = adjMSampleCell{3,1};
flevelM = adjMSampleCell{4,1};

for k = 1:length (cand_route)
 %cost_this_route = 0;
this_route = cand_route{k,1};

if (coll_enabled)
   %check if conflict detected
   [~, coll_status, ~] = rep_collision1a(this_route, start_time, adjMg,...
                       wpt_status2, time_spacing2, start_wp, dest_wp, spd2);
   coll_detected = coll_status;
   %penalize traffic-impacted route
   if (coll_detected)
      cost_route3(k,:)= Inf ;
      continue %very important to prevent overwriting penalty costs
   end
```

```
end

if length(this_route)<=1||this_route(1)~=start_wp ...
            || this_route(length(this_route))~=dest_wp,
           flt_dist = Inf; wea_cost_this2 = Inf;
           level_cost = Inf; delta_rt= Inf;
           cost_route3(k,:)= Inf ;
           continue;
else
%calculate fuel and flight level costs if weighting is not zero
     flt_dist=0; level_cost=0;

   for n = 1: length (this_route)-1,
      wp_id =this_route(n);
      next_wp_id = this_route(n+1);
    if wb ~= 0
      %calculate distance
%        if exp_arr_time == 0 %indicates that actual flight time was required
      next_dist = adjMg (wp_id, next_wp_id);
         if next_dist == 0
            flt_dist=Inf;
            level_cost = Inf;
         end
      flt_dist = flt_dist + next_dist;
     %        end
    end
   %level change costs
   if w1 ~= 0 && next_dist~= 0
      wpt_level_cost = level_chg_cost*abs(flevelM (wp_id,2) - flevelM (next_wp_id,2));
    %check if not starting with start node and end with destination
      level_cost = level_cost + wpt_level_cost';
   end
   end

%get fuel costs for the route
         cost_route3(k,5) = wb*burn_coeff2*(flt_dist-deft_dist2);

    if w1== 0
       cost_route3(k,4) =0;
    else
    %get level costs for the route
      cost_route3(k,4) = w1*level_cost;
    end

%calculate expected times if needing delay and weather costs
    if wdelta == 0
      cost_route3(k,1) = 0;
    end
    if wdelta ~= 0 || wm ~=0
     % actual_arv_time = start_time+flt_time/60;
      %flt_time = cost_route(k,1)/(wdelta*Cdelta)
     wpt_timing2 = get_est_time_wpt (this_route, adjMg, start_time, spd2);
    % if ~isempty(wpt_timing2)
     if size(wpt_timing2,1)>1
         actual_arv_time = wpt_timing2(length(wpt_timing2),1);
     else
        actual_arv_time = Inf; %arbitarily large(tends to infinity)
     end
      delta_rt = actual_arv_time - exp_arr_time;
      %pause
      %set to zero if negative (early arrival)
```

```
        if delta_rt<0,
            delta_rt=0;
        end
        if wdelta ==0
            cost_route3(k,1) = 0;
        else
        cost_route3(k,1) = wdelta*delta_rt*Cdelta;
        end
     end

%calculate weather impact costs
 if wm == 0
       cost_route3(k,2) = 0;
 else
    adjMg2 = adjMSampleCell{2,1};
    cost_this_route2 = 0; next_link_cost2=0;
    for m = 1: length (this_route)-1,
        wp_id =this_route(m);
        next_wp_id = this_route(m+1);
        next_link_cost2 = adjMg2 (wp_id, next_wp_id);
        cost_this_route2 = cost_this_route2 + next_link_cost2;
    end

%calculate costs for moving weather
%alternative is to update matrix
    adjMg5 = adjMSampleCell{5,1};

    wea_cost_this2 = 0; wea_wpt_cost=0;
    for m = 1: length (this_route)-1,
        wp_id =this_route(m);
        next_wp_id = this_route(m+1);
        %calculate expected times if not already calculated
        %wpt_time = wpt_timing2 {m}
        time_at_wpt =  wpt_timing2 (m,1);

        adjMSampleMovWArr = adjMg5{wp_id,2};
        for w = 1: size (adjMSampleMovWArr,1)
            start_wea_time = adjMSampleMovWArr(w,1);
            end_wea_time = adjMSampleMovWArr(w,2);
           %if (abs(time_at_wpt - start_wea_time)<= min_wea_sep)&&...
           if (time_at_wpt>=start_wea_time-min_wea_sep)&&...
            (time_at_wpt<=end_wea_time+min_wea_sep)
              wea_wpt_cost = adjMSampleMovWArr(w,3);
           end
        end
        wea_cost_this2 = wea_cost_this2 + wea_wpt_cost;

    end
        %convert cost of route to decimal
       cost_route3(k,2) = wm*wea_cost_this2/100*Cm;
 end
        %cost_route(k,2) = wm*wea_cost_this2/100*Cm

%calculate misssed connection costs
 if wMc == 0
       cost_route3(k,3) = 0;
 else
 %check if connection missed
     if next_cxn_time < actual_arv_time+min_cxn_time,
         %actual_arv_time+min_cxn_time
         %rt = this_route
```

```
      num_missed_cxn =1;
      cost_route3(k,3) = wMc*num_missed_cxn*CMc;
    else
      cost_route3(k,3) = 0;
    end
 end
end


end
```

## F. Selection and crossover functions

```
function child_route = selection_2a(parents, route_cost,cros_prob, selection_type)

num_of_pop =length(parents);

%prepare for roulette wheel selection
[cost_no_index, index_route]= sort (route_cost,1);
cost_cm = cumsum(cost_no_index);
cost_indexed = [index_route cost_no_index cost_cm];

%remove Inf
cost_cm_no = cost_cm(cost_cm~=Inf);
min_cm_cost = min(cost_cm_no);
max_cm_cost = max(cost_cm_no);
%select_pt = ceil(rand (max(cost_cm(cost_cm~=Inf))))
%%%
sel_parents(1,1)=0;
%sel_parents(1,1)=0;

switch selection_type
  case 'roulette'
    for c = 1:num_of_pop/2
    %roulette selection
    %select parent1
    select_pt = min_cm_cost + (max_cm_cost-min_cm_cost)*rand;
    for i = 1: length(cost_indexed)
        if cost_indexed(i,3) >= select_pt
       s_parent_id = cost_indexed(i,1);
       sel_parents (c,1)  = s_parent_id;
         break;
        end
    end

    % select parent2
    select_pt2 = min_cm_cost + (max_cm_cost-min_cm_cost)*rand;
    for i = 1: length(cost_indexed)
        if cost_indexed(i,3) >= select_pt2;
            s_parent_id = cost_indexed(i,1);
            sel_parents (c,2) = s_parent_id;
            %parent2 = parents{s_parent_id};
            %parent2_p = parent2'
        break
        end
    end
    end

    case 'tournament'
```

```
    %Tournament selection
    %number of parent per tournament
    parent_tourn = 2;
    %create an array of parent id's
    %lgt = length(parents);
    parent_id_array = (1:num_of_pop);
    parent_won = [];
    %get set of parent1

    %%%%%%%%%%%%

     for c=1:num_of_pop
         % if c>=num_of_pop
             num_parent_remain = length(parent_id_array);
       if num_parent_remain<=2
           break
       else
           sel_two = randperm (num_parent_remain,2);
           %sel_two = randperm (num_of_pop,2);
           first_select = sel_two(1,1);
           second_select = sel_two(1,2);

           if route_cost (first_select) > route_cost (second_select)...
              || (route_cost (first_select)== route_cost (second_select)...
               && rand<=0.5);
            parent_won (c,1) = parent_id_array(1,first_select);
            parent_id_array = [parent_id_array(1:(first_select-1)), ...
              parent_id_array((first_select+1): length(parent_id_array))];
           else
           %('SECOND PARENT WON')
            parent_won (c,1) = parent_id_array(1,second_select);
            parent_id_array = [parent_id_array(1:(second_select-1)), ...
            parent_id_array((second_select+1): length(parent_id_array))];
           end
           %parent_id_array2 = parent_id_array
           %pause
       end
     end

    for j=1:num_of_pop/2
      par_id = randperm (length(parent_won),2);
      sel_parents (j,1:2)= [parent_won(par_id(1,1)), parent_won(par_id(1,2))];
    end

    otherwise
    disp('Unknown selection type')
end

if length(sel_parents)>1
child_route = crossoverb (sel_parents, parents, cros_prob);
else
 %return original set
 child_route = parents;
end

function child_route = crossoverb (sel_parents, parents, cros_prob)
child_route = cell (length (parents),1);

for n = 1:length (sel_parents)

parent1 = [];
```

```
parent2 = [];
    %get parent route from id's
id_parent1 = sel_parents(n,1);
id_parent2 = sel_parents(n,2);
    if (id_parent1&& id_parent2) > 0

    parent1 = parents{id_parent1};
    parent2 = parents{id_parent2};
    %else continue
    end

k=0; % reset index of common wp cell array
%common_wpC = {};
if length(parent1)>length(parent2)
    common_wpC = cell(length(parent1),1);
else
    common_wpC = cell(length(parent2),1);
end

if  rand <= cros_prob && ~isempty(parent1)&&~isempty(parent2)
%remove start and destination - get common
for i=2: length(parent1)-1
    for j=2 : length(parent2)-1
    if parent1(i,1) == parent2(j,1)
        k=k+1;
        common_wpC {k}= [i,j];
        %common_wpC_k =  common_wpC {k};
    end
    end
end

if ~isempty(common_wpC)&& k~=0
   %select a common point randomly
   cp = ceil(rand*(length(common_wpC)));
    common_ptM = common_wpC {cp};
    if ~isempty(common_ptM)
   crs_pt1 = common_ptM(1,1);
   crs_pt2 = common_ptM (1,2);
   %swap the sections
    child1= [parent1(1:crs_pt1);parent2(crs_pt2+1:length(parent2))];
    child2= [parent2(1:crs_pt2);parent1(crs_pt1+1:length(parent1))];
    child_route {2*n-1,1} =  child1;
    %child_route_p1 = child_route {2*n-1,1}';
    child_route {2*n,1} = child2;
    else
    child_route {2*n-1,1} = parents{sel_parents(n,1)};
    child_route {2*n,1} = parents{sel_parents(n,2)};
    end
 else
    child_route {2*n-1,1} = parents{sel_parents(n,1)};
    child_route {2*n,1} = parents{sel_parents(n,2)};
end

else
    child_route {2*n-1,1} = parents{sel_parents(n,1)};
    child_route {2*n,1} = parents{sel_parents(n,2)};
end

end
```

## G. Initialisation of population

```
function pop_cand_route =  g_init_pop_quo(start_wp, dest_wp, adjMSample, num_of_pop)
% BS AYO 17/08/16

max_attempts=1000; attempts=1;

%get initial path
current_path =[];
[current_path, cost_r] = get_cand_route_quo(start_wp, dest_wp, adjMSample);
pop_cand_route{1,1} = current_path;
p=1;y=0;
while p <= num_of_pop && attempts<=max_attempts,
   %check if route aready in population
  [cand_r, cost_r] = get_cand_route_quo(start_wp, dest_wp,adjMSample);

  if ~isempty(cand_r)
     pop_cand_route{p,1} = cand_r;
     p=p+1;
  end
attempts = attempts+1;
end

%%%%%%%%%%

function [cand_route, cost_route] = get_cand_route_quo(start_wp, dest_wp,adjMSample)

next_wp = start_wp;
previous_wp =  start_wp;
cand_route (1,1)= next_wp;
cost_route=0;
 %checks if waypoint is to be included
len_AdjM = length(adjMSample);

max_attempts = 500;
% for loop section to generate one candidate route
wp_index =1; num_attempts=1;


while next_wp ~= dest_wp && num_attempts<max_attempts,

   %%get list of adjacent wp
   %adj_current = (adjLcurr {previous_wp});
   %get the row in adj matrix for previous wp
   rowm = adjMSample(previous_wp,:);
   %get list of adjacent wp
   adj_current = find(rowm);

   %return empty list if route is invalid
   if isempty(adj_current)
      cand_route = [];
      break
   else
   %pick one of the waypoints randomly
   next_wp = adj_current(1,ceil(rand*(length(adj_current))));
   end
   %remove waypoint
%     adjMSample (previous_wp, :) = zero_col;
%     adjMSample (previous_wp, :) = 0;
   adjMSample (:, previous_wp) = 0;
```

```
    wp_index = wp_index+1;
    cand_route (wp_index,1)= next_wp;
    previous_wp = next_wp;

    %cand_route'
    %adj_current = (adjLcurr {next_wp});
    num_attempts= num_attempts+1;
end

end
```

## H. Visualisation functions

```
%%%%%%%%%
function rtlabel =  convert_route_id_to_labels1a (rtid,list_of_rtlabel)
% Usage: e.g rt_fixes =  convert_route_id_to_labels1a (final_result(11,1),nav_and_fixes)
rtlabel = cell(length(rtid),1);
for j = 1: length(rtid)
%rt = cell2mat(rtid{j,1});
rt = rtid(1,j);
fixes_in_rt = cell(length(rt),1);
for i = 1: length(rt)
  fixes_in_rt {i,1} =  list_of_rtlabel {rt(i)};
end
rtlabel (j,1) = fixes_in_rt;
end

%%%%%%%%%
function xyz_coord = coord3D_from_levels(coord2D, flevels)

fl = flevels;
len2D = length(coord2D);
len3D = length(fl);

xyz_coord = zeros(len3D,3);
xyz_coord(:,3) = fl(:,2); %z-coord derived from level
for i = 1: (len3D/len2D)
 xyz_coord(1+len2D*(i-1): len2D*i,1:2)= coord2D(:,1:2);
end

% line_coord_matr (xyz_ahn, adjM3D)
%%%%%%%%%
function display_network1a(drouteSet,lformat,display_param)

%set values of variables
options = display_param{1,1};
adjMSample = display_param{2,1};
coord = display_param{3,1};
traj_path = display_param{4,1};
% folder_name = display_param{5,1};
line_wgt = display_param{6,1};
algo_name = display_param{7,1};
disp_options2 = display_param{8,1};
deft_path = display_param{9,1};
draw_all_paths = options(1);
coll_enabled = options(3);
gmaps_enabled = options(4);
threeD = options(5);
```

```
    if coll_enabled == true
        for pth = 1:length(traj_path)
        line_connect_wp3a (coord, traj_path{pth,1},'k', ':', line_wgt);

        end
    end
    %draw and save path
    grid on
    if ~threeD
        hold on
%         draw_2D_graph1a(adjMSample,coord, disp_options2);
%         grid off; axis off; set(gcf,'Color','white');
    end
    plt = 1;

    for m = 1: length(drouteSet)
        droute = drouteSet{m};
        if size(droute,2)>1
        this_lformat = lformat{m};
        rt_color = this_lformat(1,3);
        rt_style = this_lformat(1,1:2);
        h1(plt) = line_connect_wp3a (coord, droute, rt_color, rt_style, line_wgt);
        lengend_arr(plt) = algo_name(m);
        plt = plt+1;
        end
    end

    if exist('h1')&&size(h1,2)>=1
    legend(h1,lengend_arr, 'Location', 'NorthOutside');
    legend('boxoff');

    end

   %show default path
    line_connect_wp3a (coord, deft_path{1,1},'c', '-.', 2);

%     if gmaps_enabled
%         plot_google_map
%         xlabel('Longitude(degrees)');
%         ylabel('Latitude(degrees)');
%     end
%


%%%%%%%%%%
%draw lines to show nodes connected in adjacency matrix matr
function h1 =  line_connect_wp3a (coord, route_wp, colr,line_style, line_width)
%num_nodes = size(matr,1);
len_rt = length(route_wp);
dimen = size (coord,2); %number of columns indicate if 2D or 3D
%check there is at least two waypoints
%if len_rt>2

    %grid on

    for n = 1: len_rt-1
      prev_wp = route_wp(n);
      this_wp = route_wp(n+1);
      coord_ij = [coord(prev_wp,:) ; coord(this_wp,:)];
      switch (dimen)
```

```
        case 2
          h1 = line(coord_ij(:,1), coord_ij(:,2),...
          'Color', colr, 'LineStyle',line_style,'LineWidth',line_width) ;
        case 3
          h1 = line(coord_ij(:,1), coord_ij(:,2),coord_ij(:,3), ...
          'Color', colr, 'LineStyle',line_style,'LineWidth', line_width) ;
     end

   end
%end

%%%%%%%%%%
function draw_region1a(centre_pt, width, inten)
xy_pt = centre_pt-width/2;
adjustment = 1;
low_c = 20; medium_c = 50; high_c = 100; %define value for intensity
switch(inten) %get color attached to the intensity
   case low_c
      col = 'green';
   case medium_c
      col = 'blue';
   case high_c
      col = 'red';
end
rectangle('Position',[xy_pt,width,width*adjustment],'Curvature',[1,1],...
        'FaceColor',col);
```